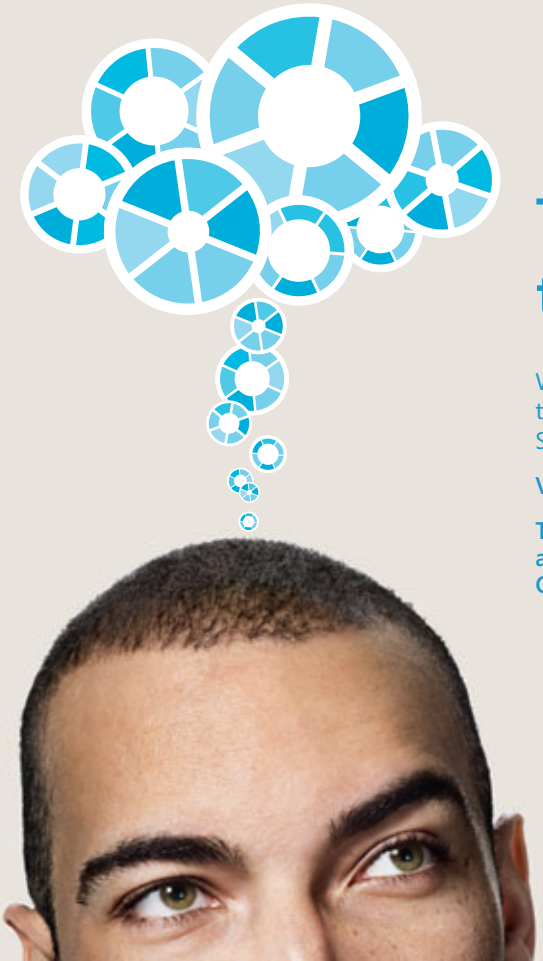
Microsoft



# The collaborative mind thinks in Windows Azure.

Windows Azure is the cloud-based development platform that's creating a new mindset for developers with unlimited capabilities. Start thinking in Windows Azure.

**Visit our stand at: TLC - COS**

**Test your coding skills. Take our weekly challenge at mobile.microsoft.com/cloud/windowsazure Or snap this tag.**

Windows Azure

THE MICROSOFT JOURNAL FOR DEVELOPERS

MAY 2011 VOL 26 NO 5

msdn magazine



## COLUMNS



Microsoft

# DESIGN DEVELOP EXPERIENCE

Quince

Using Quince™, you and your team can collaborate on the user interface using wireframes, designs and examples.

NetAdvantage®
for Silverlight Data Visualization
for WPF Data Visualization

Then use NetAdvantage® UI controls, like the map control used here, to bring the application to life quickly & easily.

msdn magazine

MAY 2011 VOLUME 26 NUMBER 5



*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

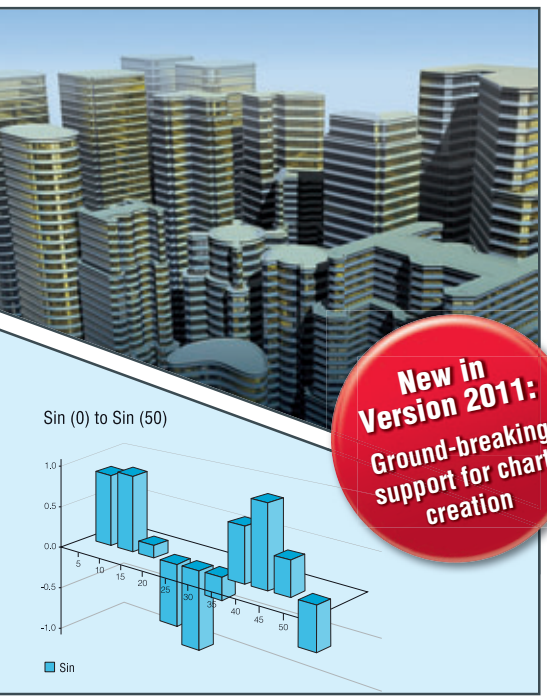
BPA WORLDWIDE

Printed in the USA

# Create cutting edge reports with the complete set of tools from Altova®

ALTOVA® missionkit®

**Experience how the Altova MissionKit®, the integrated suite of XML, database, and data integration tools, lets you display and analyze data through enhanced chart and report generation.**




## Report generation is finally easy – and affordable – with Altova MissionKit reporting tools:

**StyleVision®** – stylesheet and report design tool

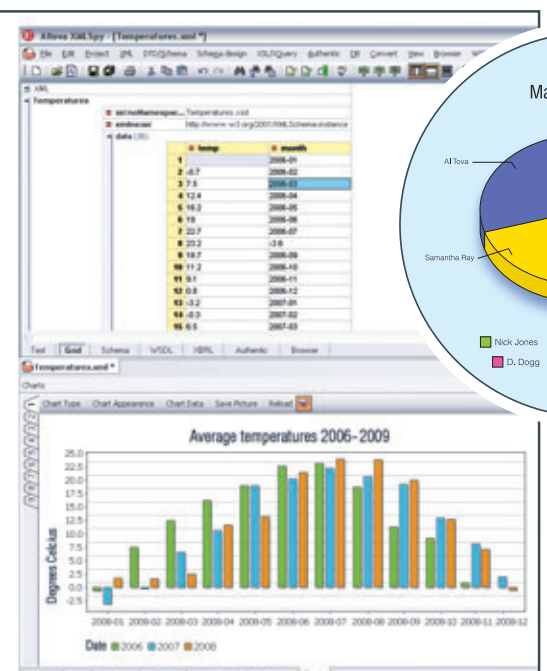
- Consolidated reporting based on XML, database and/or XBRL data
- Dynamic data selection and rendering
- HTML, Microsoft Word, PDF, and e-Form report creation

**XMLSpy®** – advanced XML editor

- Instant creation of charts to analyze XML data
- One-click export of charts to XSLT, XQuery, or image files

**MapForce®** – any-to-any data mapping tool

- Integration with StyleVision for rendering attractive reports
- Report generation for virtually any data format: XML, databases, EDI, flat files, Excel 2007+, and more




Download a 30 day free trial!

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

# EDITOR'S NOTE

KEITH WARD

# Staying Put

I was on a flight to Redmond recently for some meetings on The Campus. The man in the seat next to me was playing Solitaire on his iPad. Tucked away under the seat in front of him was a fat laptop bag. Because he looked like a standard-issue businessman, I struck up a conversation about how he uses his iPad versus his laptop. Specifically, I wanted to see if his tablet has replaced his laptop as his primary business computer.

The iPad hasn't replaced his laptop, he said, although he enjoys using it immensely. Frankly, it was the answer I expected. It's also the reason that—contrary to growing public perception—laptop and desktop computers are not on their way out. They don't have one foot in the grave and the other on a banana peel, soon to be replaced by iPads, Motorola Xoom tablets, Androids, iPhones and, yes, even Windows Phone 7 smartphones.

And that fact is why developers who toil away making applications for desktop OSes like Windows 7 and the Mac need not worry about job security. You're not Sheriff Woody, knocked off the bed by "Shiny New Toy" Buzz Lightyear and about to be put in a yard sale box.

There's no doubt that mobile OS development is all the rage right now. There's nothing wrong with that. At one time, Java was all the rage. C++ was all the rage. ASP.NET was all the rage. You get the point. What I'm saying is not that those languages are dead now, so it's time to panic if you're not versed in the intricacies of mobile apps. On the contrary: All those languages are still going strong, and developers are still building great software with them. Thus shall it be with more traditional OS app development: Your niche isn't going away. Heck, it's not even a niche. There are a lot of PCs out there, and they're not anywhere near extinct.

I say this with confidence for one big reason: I can't get any real work done on a smartphone, or a tablet with a smartphone OS. Sure, just like you I check e-mail and use the calendar and such, but that's about it. No one with any sense at all would try to write a document on an Android phone, for instance. That PowerPoint you need to set up for a meeting? Yeah, you could technically do it, to a limited degree, on an iPhone. In an emergency, it could work for you. But would you, would I, would *anyone* regularly do that? Come on.

The same issue afflicts tablets. Most owners that I know use them almost solely as gaming/social networking/Web browsing machines. The OSes that run on them are meant for that kind of light fiddling around, not serious work. Again, yes, you can add keyboards, stands and so on, and use them more like a laptop. But if you're doing that, then you're not really using a tablet anymore, are you?

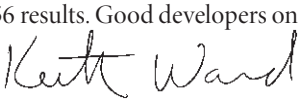
I can't see them working in a school setting, either. The lack of a physical keyboard means they'd be nearly impossible to take notes on. Forget about writing long papers on an iPhone or iPad; they're just not made for it.

You're not Sheriff Woody, knocked off the bed by "Shiny New Toy" Buzz Lightyear and about to be put in a yard sale box.

I have a Dell Studio XPS laptop as my main work machine, and an ASUS Eee PC netbook as my carry-around companion. I love my Eee PC, which I've outfitted with Windows 7 Home Premium, upgraded RAM (2GB) and ReadyBoost technology (way, way cool, by the way: tinyurl.com/49uf3zd) to get even more performance. With this setup I can write, edit, use Adobe Acrobat Pro and more—in other words, almost all the work I need to get done, I can do on my three-pound, not-much-bigger-than-an-iPad netbook.

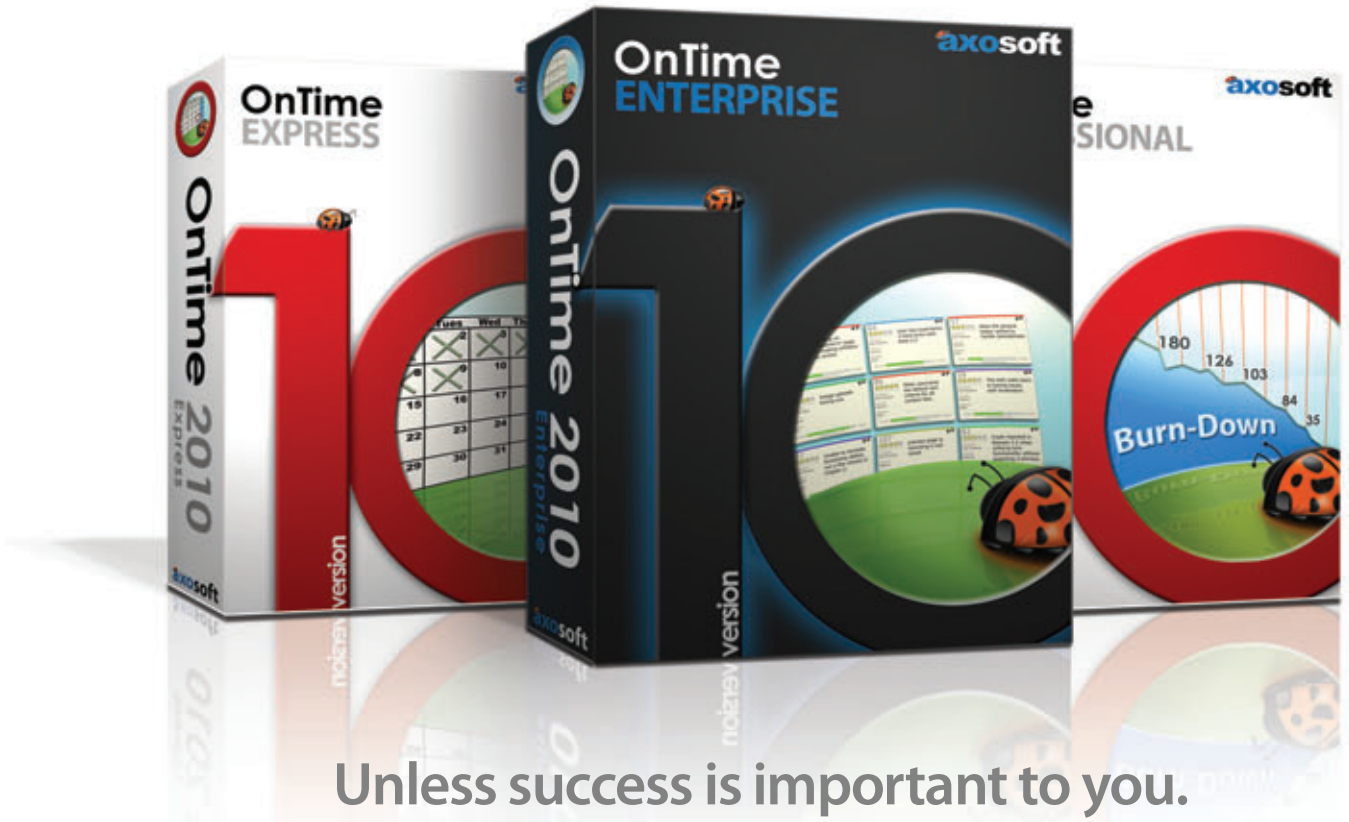
Those advantages apply to business as well. There's simply no substitute for a full-featured OS when it comes to getting real work accomplished. This isn't to imply that smartphones and tablets are useless—only that it's critical to know what they can, and can't, do. And what they can't do will keep the trusty old PC around for many years.

Certainly, there's nothing wrong with old dogs learning new tricks. But if you like what you're doing now and don't feel like squeezing into the fast lane, take heart: On March 24, a Dice.com job search using the term "Fortran" returned 56 results. Good developers on any platform, even one older than most of you, will find work.



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

# It's not really that helpful...




## Unless success is important to you.

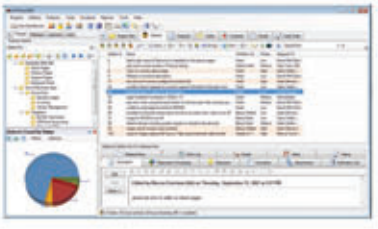
Project management and bug tracking for Agile & Scrum dev teams.

### Hosted

OnTime Now!

**OnTime Now!** Managing an Agile or Scrum dev team using a cloud-based solution has never been easier or more cost effective. Start a Free 30-Day Trial in seconds and see why Axosoft is a leading provider.

### Windows



**OnTime for Windows** is installed in your own server environment. It uses a .NET & SQL back-end, integrates easily with Visual Studio, and is a total joy to use in the rich Windows client. Includes free SDK / APIs.

### Web

**OnTime Web** provides you with an anywhere-anytime solution, whether you go with OnTime hosted or installed. This is what a web app is supposed to be like -- loaded with features and fully capable.

Free 1-user License • Free Download • Free Team Trial • Free Web Demo

Some of our awards:

axosoft.com
SOFTWARE FOR SOFTWARE DEVELOPMENT™
800.653.0024



DINO ESPOSITO

# Code Contracts Settings in Visual Studio 2010

Last month I introduced software contracts as they're implemented in the Microsoft .NET Framework 4. Known as Code Contracts, software contracts allow you to express formal conditions your code should fulfill to work properly. Code Contracts cause your code to throw an exception if a method doesn't receive data as expected or if it produces data not following expected postconditions. For an overview of preconditions and postconditions, you might want to check out my article in last month's issue (msdn.microsoft.com/magazine/gg983479).

Code Contracts are part of the .NET Framework 4, but also rely on a few facilities in Visual Studio 2010, such as runtime tools, integration with MSBuild and a property page in the Project Properties box. It's important to note that just writing preconditions and postconditions is not enough. You also need to turn on runtime checking capabilities for each project to enjoy software contracts. You do that through the Code Contracts project property page in Visual Studio 2010.

In this article, I'll discuss the intended purpose of the various options you can check or select and drill down into the rewriter tool and practices for the most common thing you do with Code Contracts: argument validation.

## The Code Contracts Property Page

Should Code Contract preconditions and postconditions be enforced in all builds or only in debug builds? In practice it comes down to your conception of a software contract. Is it part of the design effort? Or is it just a debugging measure?

If it's a design function, then there's no reason to strip out Contracts in release builds. If it's just a debugging technique, then you don't want to have it around when you compile in release mode.

In the .NET Framework, Code Contracts are just a part of the framework and are not baked into any of the languages. As a result, it's easier to configure them on a per-build basis within your project. The .NET Framework implementation of software contracts therefore leaves it up to you to decide where and when it's appropriate to implement the Contracts.

**Figure 1** shows the property page in Visual Studio 2010 through which you set up how you want software contracts to work for an application. Note that such settings apply on a per-project basis and therefore can be adapted as appropriate.

You can select the configuration of choice (Debug, Release and so on) and apply settings only to that configuration. In this way, you can enable Code Contracts for debug, but not for release and, more importantly, you can reverse the decision at any time.

## Runtime Checking

To enable Code Contracts you must check the Perform Runtime Contract Checking option. If you leave that unchecked, then any Contract instructions you may have around the source code will produce no effect (except for Contract.Assert and Contract.Assume in any build where the DEBUG symbol is defined, but that's sort of a minor point). The checkbox controls whether the rewriter tool is triggered at the end of each compile step. The rewriter is an external tool that post-processes software contracts and modifies the MSIL code, placing precondition, postcondition and invariant checks in the right places.

Note, however, that if you have a precondition like this, you're going to get a runtime assert failure if you keep the rewriter off:

```
Contract.Requires<TException>(condition)
```

**Figure 2** shows the message box you get.

> Code Contracts are part of the .NET Framework 4, but also rely on a few facilities in Visual Studio 2010.

To see how runtime checking works in detail, consider the following code:

```
public Int32 Sum(Int32 x, Int32 y) {
  // Check input values
  ValidateOperands(x, y);
  ValidateResult();

  // Perform the operation
  if (x == y)
    return 2 * x;
  return x + y;
}
```

The contract details are stored in the ValidateXxx methods using ContractAbbreviators as discussed in last month's column. Here's the source code for ValidateXxx methods:

```
[ContractAbbreviator]
private void ValidateOperands(Int32 x, Int32 y) {
  Contract.Requires(x >= 0 && y >= 0);
}

[ContractAbbreviator]
private void ValidateResult() {
  Contract.Ensures(Contract.Result<Int32>() >= 0);
}
```

If you use Contract.Requires instead of Contract.Requires<TException>, then you save yourself from the failure of **Figure 2** if you keep the rewriter off in one of the builds. The message box in **Figure 2** is due to the internal implementation of Contract.Requires, which looks like this:

```
[Conditional("CONTRACTS_FULL")]
public static void Requires(bool condition, string userMessage) {
  AssertMustUseRewriter(
    ContractFailureKind.Precondition, "Requires");
}

public static void Requires<TException>(bool condition)
  where TException: Exception {
  AssertMustUseRewriter(
    ContractFailureKind.Precondition, "Requires<TException>");
}
```

The Conditional attribute indicates to compilers that such a method call should be ignored unless the CONTRACTS_FULL symbol is defined. This symbol is defined only when you turn on the Perform Runtime Contract Checking option. Because Contract.Requires<TException> is not conditionally defined and lacks the attribute, the rewriter check is performed and results in a failed assertion if runtime Contract checking is disabled.

Let's move ahead and consider the effect of using the rewriter on the previous code. You can easily verify for yourself what I'm saying by just using breakpoints and hitting Ctrl+F11 to bring up the Disassembly view in Visual Studio 2010. **Figure 3** shows the content of the Disassembly view when you step through the method Sum compiled without enabling runtime Contract checking. As you can see, the source code is the same as you wrote in the class.

When you enable runtime checking, the rewriter tool passes over the compiler returns and edits the MSIL code. If you step through the same code with Code Contracts enabled, you see something like **Figure 4**.

The notable difference is that ValidateResult is invoked right before exiting the Sum method and past the return statement. You don't have to be an MSIL guru to read what's going on in the code in **Figure 4**. Operands are validated before the method starts honoring the topmost position of preconditions. The code that contains postconditions is moved to the bottom of the method and the MSIL code for the final return statement just falls into it. More interestingly, the first return statement—the one that in the Sum method implements a shortcut—now just jumps to the address where ValidateResult begins:

```
...
          return 2 * x;
00000054  mov         eax,dword ptr [ebp-8]
00000057  add         eax,eax
00000059  mov         dword ptr [ebp-0Ch],eax
0000005c  nop
0000005d  jmp         0000006B
...
           ValidateResult();
0000006b  push        dword ptr ds:[02C32098h]
...
```

Going back to **Figure 1**, notice the drop-down list near the Perform Runtime Contract Checking checkbox. That list lets you indicate the number of software contracts you want to enable. There are various levels: Full, Pre and Post, Preconditions, ReleaseRequires and None.

Full means that all types of software contracts are supported and None means that none of them are taken into account. Pre and Post excludes invariants. Preconditions also excludes Ensures statements.

ReleaseRequires doesn't support the Contract.Requires method and only allows you to specify preconditions using Requires<TException> or the legacy If-Then-Throw format.

The Project Property page allows you to enable or disable runtime checking on a per-project basis, but what if you want to disable runtime checking only on a few sections of your code? In that case you can just disable runtime checking programmatically using the ContractRuntimeIgnored attribute. However, a more recent release (1.4.40307.0) added a new Skip Quantifiers option that also allows you to not execute any contracts that contain references to Contract.ForAll or Contract.Exists.

You can apply the attribute to members you use in Contract expressions. If the member is decorated with the attribute then

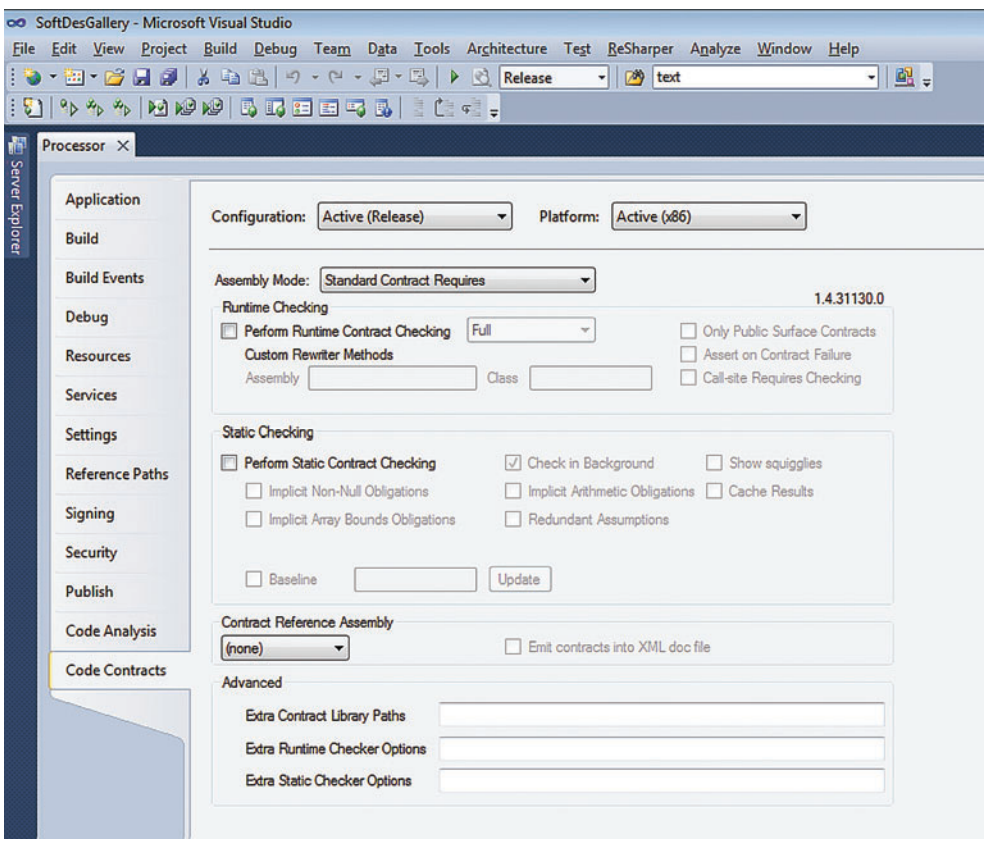



Figure 1 **The Property Page for Code Contracts in Visual Studio 2010**

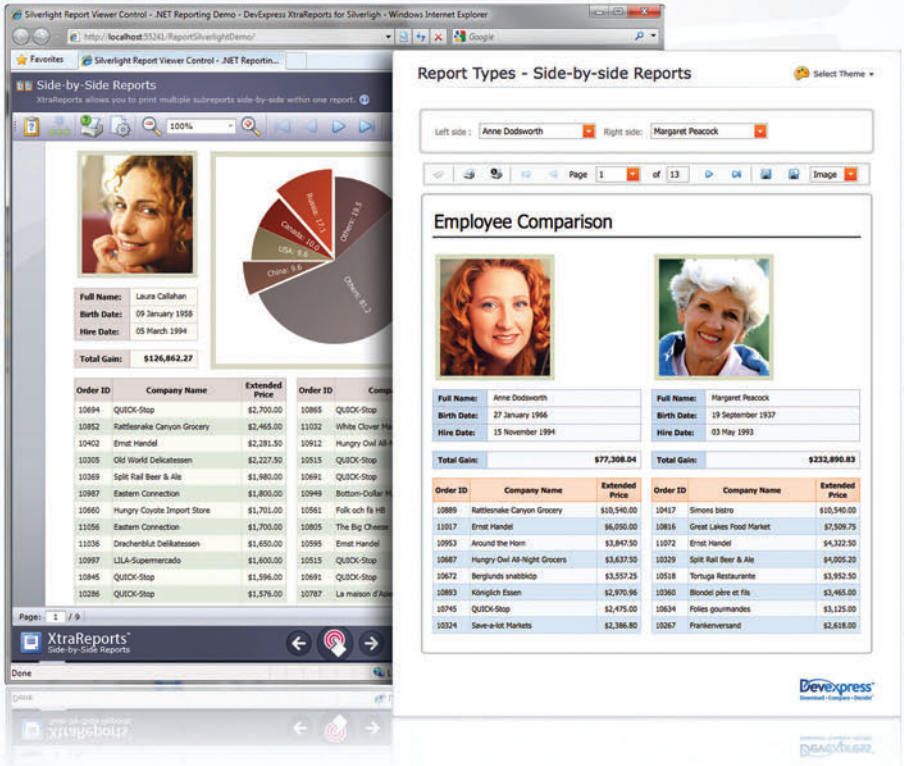
# ca·pa·ble (rə'pôrting)

**– *adjective***

1. having power and ability; efficient; competent
2. having the tools required for a specific task

**– *synonyms***

*accomplished, efficient, experienced*




A Reporting Library should be able to accommodate the structure of your data. Our banding system allows for the skillful construction of elegant reports that conform to you and your data–without imposing restrictions on your ability to address customer needs in the shortest possible time.

**Experience the power of the XtraReports Suite today.**
**Download your FREE trial – www.DevExpress.com/FreeEval.**

Devexpress™
Download • Compare • Decide!

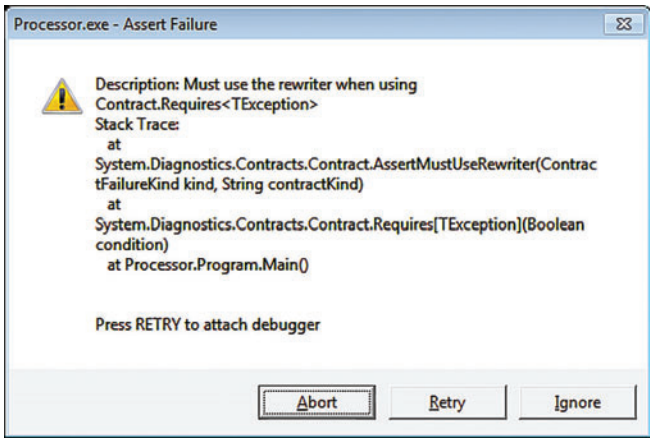



Figure 2 **The Code Requires Runtime Contract Checking**

the entire Contract statement in which it appears is not subjected to runtime checking. The attribute is not recognized in Contract methods such as Assert and Assume.

## Assembly Mode

The Code Contracts properties also let you configure an Assembly Mode setting for the Contracts. The setting refers to how you intend to perform argument validation. There are two possible options: Standard Contract Requires and Contract Reference Assembly. The Assembly Mode settings help tools like the rewriter to fine-tune the code and give proper warnings when necessary. Let's say that you use the Assembly Mode to declare your intended use of Code Contracts for parameter validation. The Assembly Mode settings introduce a couple of simple rules you must fulfill, otherwise you'll get a compile error.

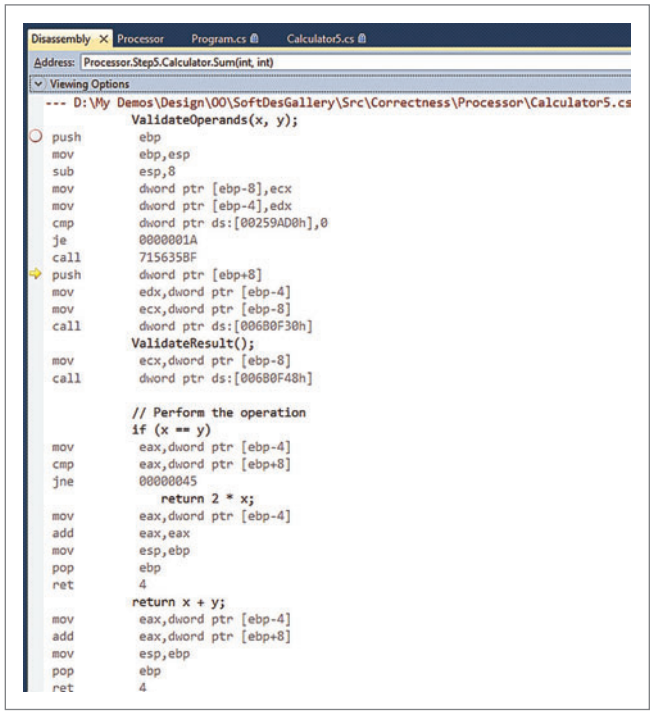



Figure 3 **Disassembly View Without Runtime Contract Checking**

Assembly Mode should be set to Standard Contract Requires if you use methods Requires and Requires<T> to validate method arguments. You should use Custom Parameter Validation if you use any If-Then-Throw statement as preconditions. If you don't use Custom Parameter Validation, the statement will be treated like a Requires<T>. The combination of Custom Parameter Validation and explicit use of any form of Requires statements will, instead, throw a compiler error.

What's the difference between using Requires and using If-Then-Throw statements? An If-Then-Throw statement always throws the exception you indicate if validation fails. In this regard, it differs from Requires, but it's similar to Requires<T>. A plain If-Then-Throw statement is also not discoverable by Contract tools (rewriter and static checker) unless you follow it with a call to EndContractBlock. When you use EndContractBlock, it must be the last Code Contract method you invoke in the method. No other Code Contracts call can ever follow it:

```
if (y == 0)
  throw new ArgumentException();
Contract.EndContractBlock();
```

> If you want to throw specific exceptions at run time, then using Requires<TException> is the way to go.

In addition, Requires statements are automatically inherited. An If-Then-Throw statement is not inherited unless you also use EndContractBlock. In legacy mode, If-Then-Throw Contracts are not inherited. Instead you must manually do the Contract inheritance. The tools will try to warn if they do not detect that the preconditions are repeated in overrides and interface implementations.

Finally, note that ContractAbbreviators are not allowed to contain any If-Then-Throw statements, but you can use Contract validators for that. Abbreviators can only include regular Contract statements for argument validation.

## Other Settings

In the Code Contracts property page, if you select the Perform Runtime Contract Checking option, you'll enable some additional useful options.

If you turn on the Assert on Contract Failure option, any violations of a contract will result in an assert that describes the context of the failure. You'll see a message box similar to what's shown in **Figure 2** and will be given a few options to choose from. You can, for example, try to attach a debugger again, abort the application or simply ignore the failure and proceed.

You might want to use this option for debug builds only because the information displayed is not likely to be meaningful to the average end user. The Code Contracts API offers a centralized exception handler that captures any violation, leaving it up to you to figure out what exactly went wrong. Information you receive distinguishes whether a precondition, a postcondition or invariant

# flex·i·ble (rə'pôrting)

**– *adjective***

1. easily modified; adaptable
2. demonstrating the ability to adapt to new and different requirements

**– *synonyms***

*adaptable, modifiable, adjustable*


A Reporting Library must be flexible so as to fully accommodate your platform requirements. The XtraReports Suite allows you to target WinForms, ASP.NET, WPF or Silverlight–without the need to re-create individual reports from scratch.

**Experience the power of the XtraReports Suite today.**
**Download your FREE trial – www.DevExpress.com/FreeEval.**

DevExpress™
Download • Compare • Decide!

failed, but only uses the Boolean expression and possibly the configured error message to characterize the error. In other words, it's a bit difficult for you to recover gracefully from the centralized exception handler:

```
Contract.ContractFailed += CentralizedErrorHandler;
```

Here's some code that illustrates the handler:

```
static void CentralizedErrorHandler(
  Object sender, ContractFailedEventArgs e) {
  Console.WriteLine("{0}: {1}; {2}", e.
    FailureKind, e.Condition, e.Message);
  e.SetHandled();
}
```

If you want to throw specific exceptions at run time, then using Requires<TException> is the way to go. You might want to use Requires and the centralized handler if you intend to limit the use of Contracts to debug builds or if you don't care what the actual type of the exception is. It's often enough just to indicate that an error occurred. For instance, at the top level, many applications have a catch-all that catches every type of exception and figures out how to restart.

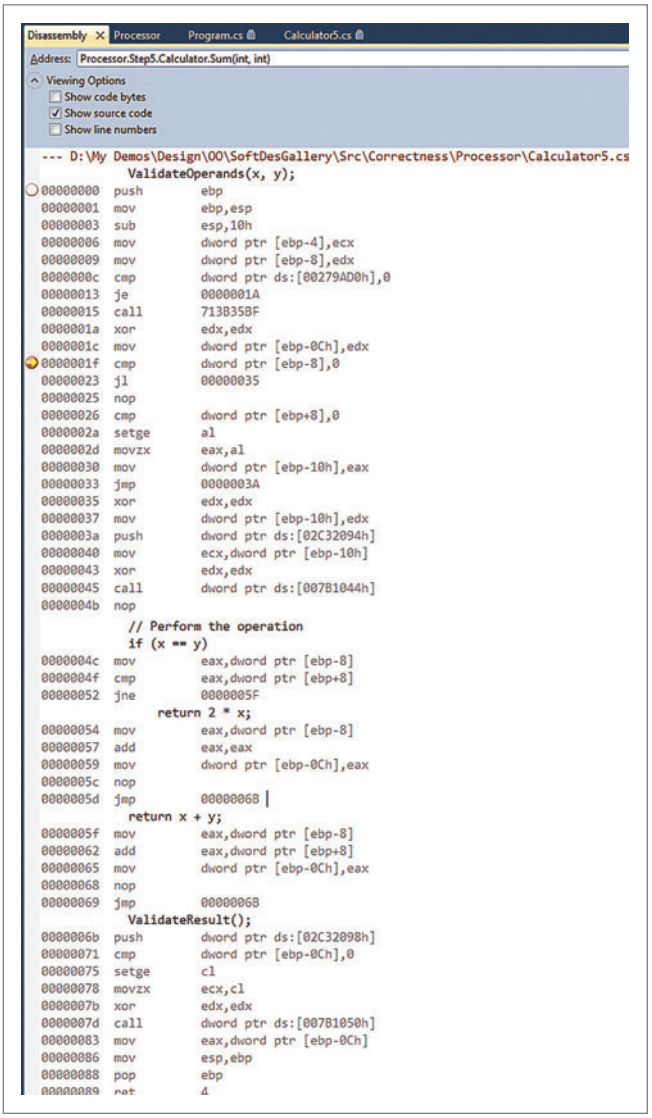



Figure 4 **Postconditions Checked Past the Return Statement**

The Only Public Surface Contract option refers to where you would like to have Code Contracts enforced: every method or only public methods. If you check the option, then the rewriter ignores Code Contract statements on private and protected members and it only processes Contracts on public members.

Checking this option makes sense if you incorporate Code Contracts as part of your overall design and, as a result, use them everywhere. However, once the application is ready to ship, as a form of optimization you can turn off the extra burden of checking parameters on internal members because no external code will ever be invoking those members directly.

Whether limiting Code Contracts to the public surface of an assembly is a good idea also depends on how you've written the code. It's a form of optimization if you can guarantee that any calls the public surface makes to lower levels are correct. If not, disabling contracts for internal methods can turn into the source of nasty bugs.

The Call-site Requires Checking option serves another optimization scenario. Suppose you're writing a library to be used by modules in other assemblies. For performance reasons, you disable runtime checking of Contracts. However, as long as you also create a Contract reference assembly, you enable the caller to check the Contract for each method being called.

A Contract reference assembly may exist for every assembly that contains classes using Code Contracts. It contains the publicly visible interface of the parent assembly with Contract statements, but no other code. The creation of the assembly can be ordered and controlled from the Code Contracts property page.

Any code that intends to invoke your library may reference your Contract reference assembly and could automatically import your Contracts by simply turning on the Call-site Requires Checking option. When processing the caller code, the rewriter will import the Contract for every method being called on an external assembly that comes with a Contract reference assembly. In this case, the Contract is checked at the call-site—on the caller side—and remains an optional behavior that can be turned on and off for code you don't control directly.

## Wrapping Up

Code Contracts is an area of the .NET Framework that's worth a lot more investigation. I've just scratched the surface of the configuration options here and haven't even gone into the use of the static checker. Code Contracts help you design your apps more clearly and write cleaner code.

To learn more about Code Contracts, see the June 2009 CLR Inside Out column by Melitta Andersen (msdn.microsoft.com/magazine/ee236408) and the DevLabs Code Contracts site (msdn.microsoft.com/devlabs/dd491992). You'll also find interesting background information about the development of Code Contracts at the Microsoft Research Code Contracts site (research.microsoft.com/projects/contracts). ■

**DINO ESPOSITO** *is the author of "Programming Microsoft ASP.NET MVC" (Microsoft Press, 2010) and coauthor of "Microsoft .NET: Architecting Applications for the Enterprise" (Microsoft Press, 2008). Based in Italy, Esposito is a frequent speaker at industry events worldwide. Follow him on Twitter at twitter.com/despos.*

**THANKS** to the following technical expert for reviewing this article: *Mike Barnett*

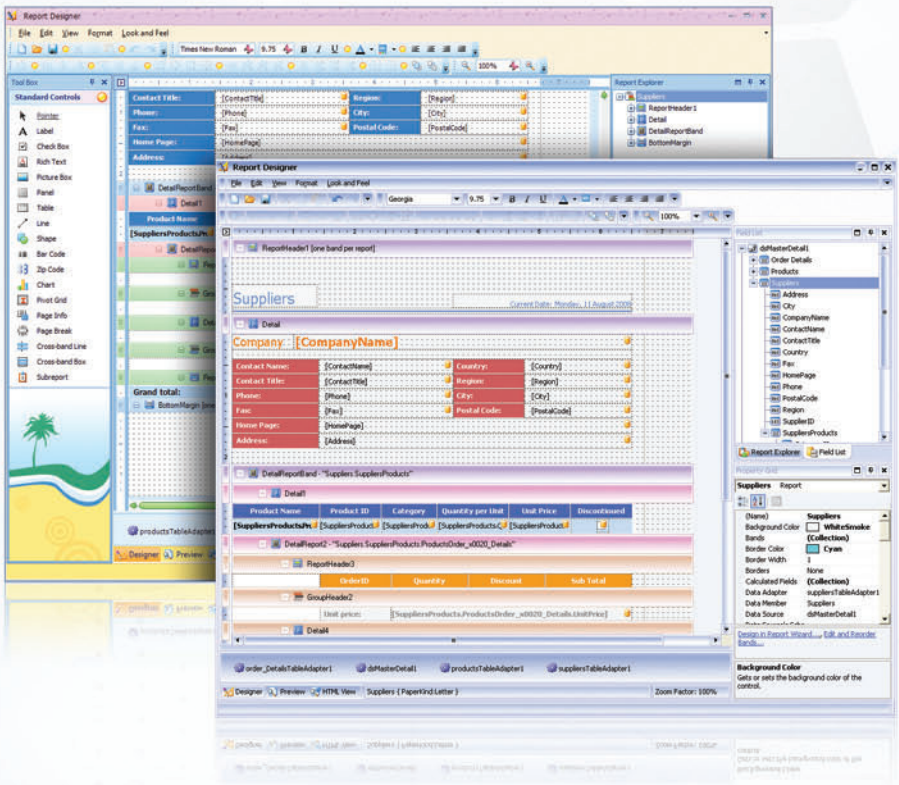
# de·pend·a·ble (rə'pôrting)

**– *adjective***

1. trustworthy; reliable
2. able to rely upon an individual or organization in order to achieve your goals

**– *synonyms***

*solid, faithful, responsible*




Though the award-winning XtraReports Suite delivers on its promise of multi-platform reporting, technical questions and issues will undoubtedly arise. Our support team is second to none when it comes to timeliness and quality–we are your extended team and want to do everything possible to ensure your success in the marketplace.

**Experience the power of the XtraReports Suite today.**
**Download your FREE trial – www.DevExpress.com/FreeEval.**



JULIE LERMAN

# Demystifying Entity Framework Strategies: Model Creation Workflow

As a data-access technology designed to meet the needs of a variety of development styles, the Microsoft Entity Framework presents developers with a lot of options. Some of these choices need to be made early on in your development cycle and will impact what options you'll have further along in your development process. In my next three columns, I'll provide high-level guidance for some of the biggest decisions you'll need to make with the Entity Framework:

- Code First, Model First or Database First workflows for creating a conceptual model
- Lazy, explicit or eager loading
- Plain Old C# Objects (POCOs) or EntityObjects
- LINQ to Entities or Entity SQL

There are many more design decisions to be made throughout your application, but these questions are the most frequently asked—and often debated—by developers as they begin to implement the Entity Framework into existing or new applications.

I'll attack the conceptual model creation workflow here and cover the other three topics in the next two columns. My goal is to provide a high-level overview.

## Three Options for Creating a Conceptual Model

The Entity Framework relies on a conceptual model of your domain entities, called an Entity Data Model (EDM), and choosing how to create that model is the first decision you'll need to make. There are three separate workflows to choose from:

- The Database First workflow begins with a legacy database and leverages a wizard to reverse-engineer that database into a conceptual model.
- The Model First workflow begins with an empty slate. You use the visual EDM Designer to design an EDM, then generate database schema from that model.
- The Code First workflow begins with classes that describe your conceptual model. There's no visual model used with Code First.

## Database First: Start with a Legacy Database

In the first iteration of the Entity Framework, we didn't have all of the options mentioned. The only way to create a model was to reverse-engineer an existing database into an EDM, which we refer to as Database First modeling (see **Figure 1**).

You've probably seen this demonstrated many times, in fact. You open the EDM Wizard, point to an existing database, select which tables, views, stored procedures and user-defined functions you'd like to be represented in the model and click the finish button. Instantly, a model is born. The Database First model starts its life as a virtual reflection of the database (or that subset of the database you selected). With no more effort, developers will already benefit from the Entity Framework. You can write strongly typed queries against this model, and the Entity Framework will execute those queries for you and materialize strongly typed objects from the results. Then, as you work with the results, the Entity Framework tracks the changes and allows you to persist them back to the database simply by calling its SaveChanges command.

That's going to be all that some developers need from the Entity Framework, but they're missing out on one of the great benefits of having the model, which is that you can make it look more like your domain—the classes and relationships that define your application—than the database, and then not have to worry about how to get from there back to your database (which the Entity Framework will continue to take care of for you). Model customization is a critical feature of the EDM that many developers overlook and don't

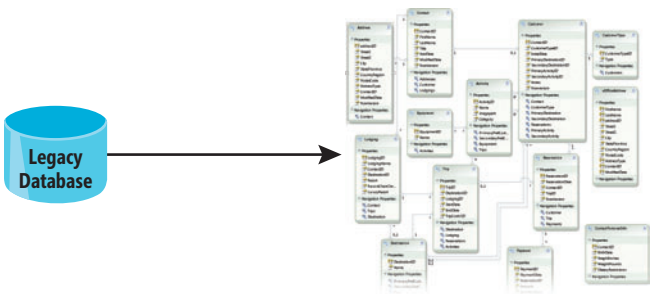



Figure 1 **With Database First, You Reverse-Engineer a Model from a Legacy Database**


Figure 2 **With Model First, You Design a Model that's Used to Generate Database Schema**

# trans·par·ent (rə'pôrting)

***– adjective***

1. free from the unknown; open; detectable
2. delivering without secrecy and vagueness

***– synonyms***

*clear, visible*


Unlike competing products which do not ship 100% of the source code when purchased, the XtraReports Suite is fully transparent and not a black box with a myriad of unknowns. When you integrate DevExpress Reporting, you can rest assured that every single line of code is available for review.

**Experience the power of the XtraReports Suite today.**
**Download your FREE trial – www.DevExpress.com/FreeEval.**

Devexpress™
Download • Compare • Decide!

Figure 3 **A Class with Code First Attributes to Ensure the Class Can Correctly Map to an Existing Table**

```
[Table("SalesOrderDetail", SchemaName="SalesLT")]
  public partial class Detail
  {
    // Scalar properties

    [Column(Name = "SalesOrderID")]
    public int OrderId { get; set; }
    [Column(Name = "SalesOrderDetailID")]
    public int DetailId { get; set; }
    public short OrderQty { get; set; }
    public int ProductId { get; set; }
    public decimal UnitPrice { get; set; }
    public decimal UnitPriceDiscount { get; set; }

    public decimal LineTotal { get; set; }
    public System.DateTime ModifiedDate { get; set; }

    // Navigation properties

    public virtual Product Product { get; set; }
    public virtual Order Order { get; set; }
  }
```

benefit from. You can introduce inheritance hierarchies into the model, reshape entities, combine and split entities, and much more.

With Database First, you can have the best of both of these worlds: leverage the existing database to get a huge head start with creating your model, then customize that model to better reflect your application domain.

## Model First: Start with a Visual Model

You won't always have a legacy database to start with. Model First lets you design the model directly in the designer and create your database schema based on the model. You can build your entities and their properties, define relationships and their constraints, and create inheritance hierarchies right in the designer. You can specify what properties will become identity keys and even if the database should be responsible for generating their values (see **Figure 2**).

The EDM Designer "Create Database from Model" feature doesn't actually create a database. What it will do is build SQL that, when executed, will define the schema of a database. This SQL is referred to as Data Definition Language, or DDL.

There are a few non-obvious things to be aware of. Model First was introduced in Visual Studio 2010 and the Microsoft .NET Framework 4. You won't find the Create Database option in Visual Studio 2008. Also, because it's a new feature, you'll need to ensure that the ADO.NET Data provider you're using (for example, System.Data.Sql) has been updated to provide this capability. The Microsoft SQL Server provider was updated for the .NET 4 release; some of the third-party providers have also been updated.

A behavior to be prepared for is that Model First does not, by default, perform incremental schema updates to your database. When you run the feature, it will create a completely new DDL script that will remove and then recreate all of the database objects. External tools are available to help you modify the database schema rather than overwrite it. Otherwise you'll want to either back up your data in advance or write a script to regenerate your test data any time you want to modify the model and recreate the database schema.

Visual Studio does provide an extensibility point in the database generation, which you can leverage with the help of the Entity Designer Database Generation Power Pack from Microsoft. You can download it from the Visual Studio 2010 Extension Manager or from visualstudiogallery.com. This extension not only provides a way for you to make incremental changes to the database with Model First, but it also gives you the ability to change the default Table Per Hierarchy inheritance mapping and even create your own customized DDL generation rules.

## Code First: Start with Code and Forego a Physical Model

With Database First and Model First, you'll end up with a physical representation of your EDM along with additional metadata. The raw format of this model is XML. It's stored in a file with an EDMX extension, and you can work with it in the EDM Designer or the raw XML. At run time, the Entity Framework reads that XML and creates an in-memory representation of the model using specialized classes that represent the metadata—entities, relationships and so on. The Entity Framework runtime works with those objects, not the actual XML file. This metadata is especially critical when the Entity Framework needs to transform model queries into database queries, database results into entity instances, and create database commands for inserts, updates and deletes. Visual Studio will generate your domain classes from the XML to use in your application.

The Entity Framework team has come up with a way to create the needed metadata objects at run time without requiring you to have a physical EDMX file. This is the power behind the third model-creation workflow for the Entity Framework, called Code First. With Code First, rather than creating an EDM, you create your domain classes as you would for any other .NET development. At run time, the Entity Framework will inspect those classes and, using a set of default conventions, build an in-memory model that the Entity Framework runtime can work with. Because your classes won't always naturally provide the information that the Entity Framework needs to create that model, you can provide additional configuration (using declarative attributes/data annotations or in code using a fluent API) to further describe the model, overriding the conventions mentioned earlier. You can use configuration for a wide variety of model tasks, from identifying primary keys that don't match Code First conventions to fine-tuning relationships or even specifying how an inheritance hierarchy should be represented.

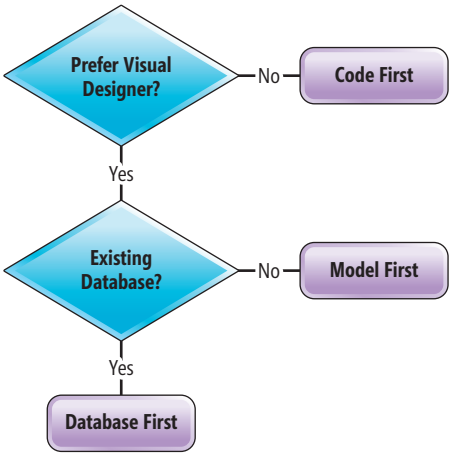



Figure 4 **The Decision Tree**

Like Model First, Code First defaults to the presumption that you aren't starting

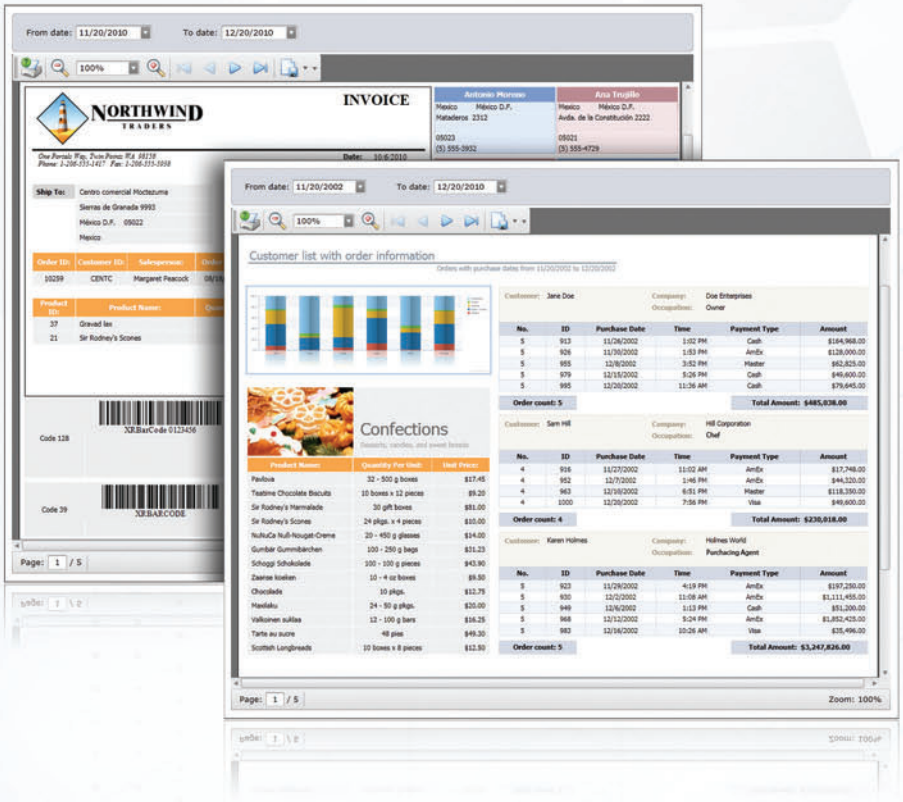
# af·ford·able (rə'pôrting)

***– adjective***

1. within a developer's financial means; reasonably priced
2. all the benefits, without the high price

***– synonyms***

*fair, reasonable*




A feature-complete and multi-platform Reporting Library should not force you to spend thousands of dollars–it should be affordable and ship with the capabilities you need to get the job done… and offer the services you need to achieve mastery. **Prices start at $349.99**

**Experience the power of the XtraReports Suite today.**
**Download your FREE trial – www.DevExpress.com/FreeEval.**

Devexpress™
Download • Compare • Decide!

with a legacy database, and provides the capability to create that database from the inferred model. Unlike Model First, you can create the database file as well as the schema. With Code First, you still lack a way to persist database data if you modify the model and regenerate the database. However, Code First does have support to detect model and database differences, as well as seed the database with data using database initializers.

You can also use Code First with an existing database. If your class and property names don't match up to the database, you can add data annotations or configuration through the fluent API to overcome that. **Figure 3** shows a class with attributes that will force Code First to provide mappings to the correct table and property names even though the class and fields don't match exactly.

## A Decision Tree

The Entity Framework has evolved to support a variety of development styles, but this now forces us to be familiar with the options in order to choose the right approach. If you like to use a UI for your model, then you have the EDM Designer available whether you're starting with an existing database or using the Model First option. Even then, you can still use the modeler to help make a stake in the ground and then use a template to create Code First classes and drop the visual model. If you're all about the code and prefer not to be tied to an EDMX file or the visual modeler, Code First will appeal to you (see **Figure 4**).

## Starting on the Right Path

Whichever modeling technique you choose—Database First, Model First or Code First—once you've created a model, you won't notice any difference when you begin to query, interact with and persist entities using the Entity Framework runtime by way of the model's classes. And it's possible to start with one workflow and switch to another. You can create Code First classes from an EDMX using the DbContext Code Generation template. And, while it's not quite as easy, you can also create an EDMX from Code First classes.

I've provided a high-level view of the choices and what might drive you to choose one over the other, and hopefully I've demystified some of the options for you.

In future columns, I'll talk about other big decisions you'll face in your Entity Framework applications, which I mentioned at the start of this column: choosing a code-generation strategy for EntityObjects or POCOs, loading related data using eager, lazy or explicit loading, and writing queries using LINQ to Entities or Entity SQL. ■

**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 the highly acclaimed book, "Programming Entity Framework" (O'Reilly Media, 2010). Follow her on Twitter at twitter.com/julielerman.*

**THANKS** to the following technical expert for reviewing this article: *Tim Laverty*



JOSEPH FULTZ

# Load Balancing Private Endpoints on Worker Roles

Early in January, David Browne and I worked on a solution to load balance internal service points on Windows Azure Worker Roles. Generally, service endpoints in Worker Roles are published, so the load balancer can take care of balancing the calls across the instances. However, the customer with whom we were working needed endpoints that were not publicly addressable. In addition, they didn't want to take on the latency of some type of queuing operation. How would we address this requirement?

During an internal event meant for us to explore various technologies and solutions, David and I came up with two different approaches for solving the challenge. For this month's column, I'll cover my design considerations and the bits of code used to prototype one of these approaches.

Not wanting to inadvertently bottleneck the final solution, we ruled out a software proxy-style solution. Instead, I chose a software mechanism that will provide a valid IP for service calls and the calling node will cache the endpoint for a given duration to reduce the overhead of endpoint resolution. The three primary strategies I considered were:

- Static assignment: assign a service endpoint to each calling node
- Centralized control: one node tracks and controls assignment of each calling node
- Cooperative control: allow any given node to indicate if it's available to service calls

Each of these choices brings with it a set of benefits and a set of disadvantages.

Static assignment has the upside of being simple to implement. If the mapping of units of work between the caller and the worker are equal, then this might be a feasible approach for balancing because the load-balancing solution for the Web Role will by extension balance the calls on the Worker Role.

The two primary disadvantages are that it doesn't address high availability for the service, nor does it address any discrepancy in load between the caller and the service node. If I attempt to morph the static assignment solution to address the problems, the solution almost assuredly starts to move toward either centralized or cooperative control.

## Centralized Control

A typical load balancer that receives health information and balances service requests based on such information utilizes centralized control. It collects information about the nodes, what it knows about the assignments it has made and any heartbeat information, and it directs requests made to the Virtual IP (VIP) to a given node.

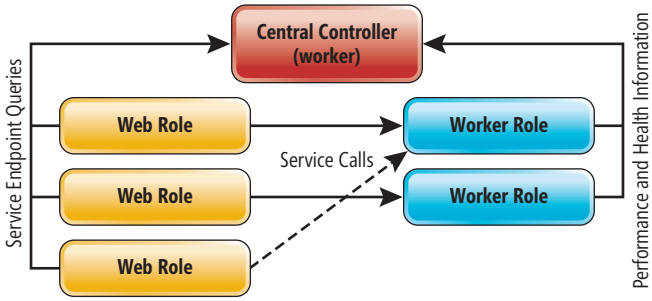



Figure 1 **Centralized Control**

In this scenario the central point would do mostly the same except it would not act as a proxy for the request, but rather the calling node will ask the central controller to receive a good address to make the call, and the controller will assign an address based on what it knows (see **Figure 1**). The calling node will cache the endpoint and use it for a predetermined quantum, which upon expiring will repeat the resolution process.

The intelligence all lies within the central controller and it must track all of the requisite information needed to determine to which node to assign calls. This could be as simple as round robin or it could take on full data collection and health analysis. It could also be complicated by the various service endpoints having different criteria to determine availability, meaning that the central controller has to be all-knowing across service implementations in the worker pool.

> If the central controller is down, then the system is down.

The biggest detractor from this implementation is that, if the central controller is down, then the system is down. This means a completely separate solution must be implemented to solve for high availability of the central controller.

In some robotic and matrix systems, worker nodes will elect a primary controller and, if the heartbeat is lost, they simply elect a new one. While this is a good design because it combines both the centralized control and cooperative control, it also adds significantly to the implementation of the load-distribution mechanism.

# ROCK STARS WANTED!

The **Visual Studio Gallery** is home to thousands of Visual Studio extensions built by rock star developers like you. Tools, controls and templates from the Visual Studio Gallery show up within the IDE and make it easy for others to create great products. Share the tools you build with ***millions of developers***.

**SHARE WHAT YOU BUILD** | visualstudiogallery.com


Extension developers, take it to the next level. The Visual Studio Industry Partner (VSIP) program helps you build, market and sell products that integrate with Visual Studio.

**BECOME A PARTNER** | msdn.com/vsip

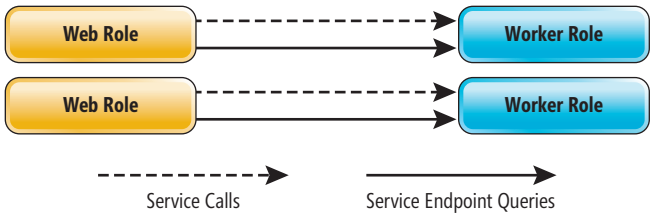



Figure 2 **Cooperative Control**

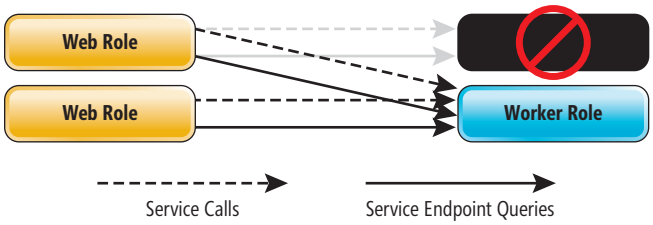



Figure 3 **Balancing to Another Node**

## Cooperative Control

Anyone who's gone through management to get someone to do something knows it can be a real hindrance to actually getting someone to do the work. Asking him directly if he has time to do it turns out to be much more expedient and, given he's a good judge of effort, is the best way to determine if he actually has time to do the work. Such is the model I followed.

The idea is that each of the calling nodes will start with its currently assigned service endpoint and ask whether it's still available (see **Figure 2**). If it isn't, the node will continue to round robin through the available pool until one responds positively (see **Figure 3**). After that, the same expiry cache mechanism described earlier is used to reduce the endpoint resolution overhead.

The upside of this design is that HA is taken care of by design and there should be high fidelity between the node determining its availability and the worker actually being able to service callers. Each of the service nodes should have the intelligence baked into its implementation that it's aware of things specific to the service that would make it available or not. This is intelligence beyond CPU and such and could be things such as availability of downstream systems that are accessed by the node. Thus, if the node returns a negative, an error or a timeout, the calling node queries the next available service node and, if available, makes its service calls to that endpoint.

The big detractor from this solution is that it requires implementation on both sides of the fence to provide an availability service and a calling protocol between the caller and the endpoints to ascertain the endpoint availability.

## The Prototype

The sample will do the following things:

- Setup a standard mechanism to determine availability
- The caller will cache an available node for a brief period
- I'll be able to disable a node for a set quantum, which should show up as all of the calls being balanced to a single node
- Once the node becomes available again, the caller should be able to return to the previous node

Some caveats: First, I'm not doing any work to intelligently determine availability, given I'm just setting up the balancing mechanism, and not worried about the intelligence behind the decision. Additionally, I'm not handling errors and timeouts, but those would be handled in the same manner as getting a negative result from the availability query. Finally, I'm simply grabbing all Worker Roles in the deployment, but in a true implementation a more intelligent way to determine all available service endpoints might be desired, such as a registry mechanism or simply attempting to hit the service on each endpoint and marking successful calls as possible endpoints. The code does go as far as to ask for a specific private endpoint and if that's different per service, then that could be used as a differentiator.

> I'll match the starting node for checking availability by using the ordinal number of the node.

The first thing to do is to do is get the list of IPs from the Worker Roles in the deployment. To accomplish that goal I have to configure the roles. For the Worker Roles I open the configuration window and add an internal service endpoints as shown in **Figure 4**.

I've also labeled the Worker Roles in the deployment as PrivateServices. Using the API of the RoleEnvironment object and the label, it's easy to fetch the nodes:

```
if (_CurrentUriString == null) {
  System.Collections.ObjectModel.ReadOnlyCollection<RoleInstance>
    ServiceInstances = null;
  System.Collections.ObjectModel.ReadOnlyCollection<RoleInstance>
    WebInstances = null;

  ServiceInstances =
    RoleEnvironment.Roles["PrivateServices"].Instances;
  WebInstances =
    RoleEnvironment.Roles["ServiceBalancingWeb"].Instances;
```

I'll match the starting node for checking availability by using the ordinal number of the node. If there are more Web Roles than

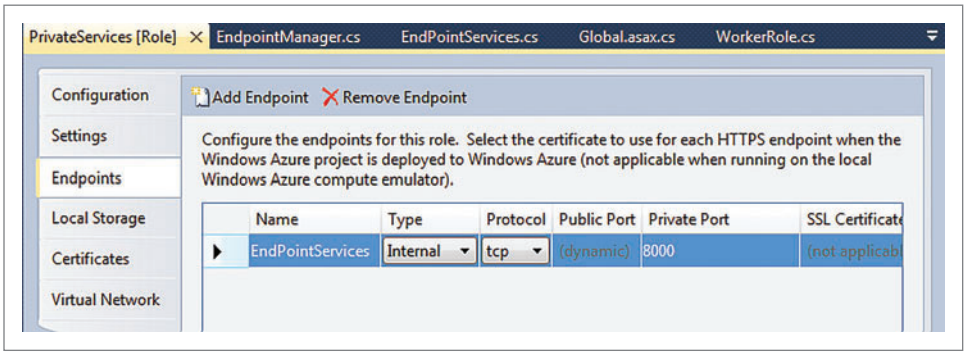



Figure 4 **Adding an Internal Service Endpoint to the Worker Role**

Figure 5 **Testing Endpoints**

```
while (!found && !Abort) {
  string testuri =
    ServiceInstances[idxSvcInstance].InstanceEndpoints[
    "EndPointServices"].IPEndpoint.ToString();
  found = CheckAvailability(testuri);
  if (found) {
    ServiceUriString = testuri;
  }
  else {
    idxSvcInstance++;
    if (idxSvcInstance >= ServiceInstances.Count) {
      idxSvcInstance = 0;
    }
    loopCounter++;
    if (loopCounter == ServiceInstances.Count) {
      Abort = true;
    }
  }
}
```

Worker Roles, I'll use a mod function to match a starting node. With the instances in hand and a starting node to test availability, I can start to loop through and test the endpoints (see **Figure 5**).

Note that there is a call to a function named CheckAvailability (see **Figure 6**). Within that function I create a binding using None for the security mode because the endpoint is internal only. I instantiate the service client and set a reasonable timeout and return the value of the call.

If an error occurs during the call, I simply return false and allow the loop to move on to the next node and check its availability. Note, however, that to determine the Web Role instance number that the code is currently executing under I've parsed the instance ID. To make this work at all I had to open an arbitrary internal (could've been external) endpoint. If I hadn't, it wouldn't increment the ID and the parse would be useless because every node would look like the only one.

Another way to create a list of nodes would be to iterate through all of the nodes, identifying the ordinal position of the current executing node in the list or to just order them by the last octet of the IP. Either of the latter two methods would be a little more foolproof, but for this particular example I just used the instance ID.

If an error occurs during the call, I simply return false and allow the loop to move on to the next node and check its availability.

One more caveat is that the structure of the ID differs between the actual deployment and the development fabric, thus forcing me to handle it in the parse code, like so:

```
string[] IdArray =
  RoleEnvironment.CurrentRoleInstance.Id.Split('.');
int idxWebInstance = 0;
if (!int.TryParse((IdArray[IdArray.Length - 1]),
  out idxWebInstance)) {
  IdArray = RoleEnvironment.CurrentRoleInstance.Id.Split('_');
  idxWebInstance = int.Parse((IdArray[IdArray.Length - 1]));
}
```

This should return a good endpoint IP that I can cache in a static variable. I then set a timer. When the time event fires I'll set the endpoint to null, causing the code to once again look for a valid endpoint to use for services:

```
System.Timers.Timer invalidateTimer =
  new System.Timers.Timer(5000);
invalidateTimer.Elapsed += (sender, e) =>
  _CurrentUriString = null;
invalidateTimer.Start();
```

Here I used a short duration of 5 seconds because I want to ensure that in a short test execution I can bounce at least one Web Role to another endpoint once I disable one of the service nodes.

## Running the Demo

Now, I'm going to modify the default page and its codebehind to simply show the node to which it established an affinity. I'll also add a button to disable a node. Both pieces of code are pretty simple. For example, the disable button will disable the service endpoint associated with the Web page to which the request gets balanced. So, it can lead to a little bit of a quirky UI feel for this test sample.

I'll add a label and a command button to the UI. In the label I'll print out the ID of the assigned endpoint and the button will allow me to disable a node so I can see all Web Roles associated with a single endpoint until the node comes back online. Inside of the code behind I add a little code on the page load to get the endpoint (see **Figure 7**).

Because I'mreally only attempting to illustrate the cooperative balancing, I haven't implemented another service method or interface, so I simply reuse the IsAvailable method to illustrate the point.

**Figure 8** shows the prototype application in action. First you can see the ID (this one is from the development fabric), the IP and whether it's available. Refreshing the page causes the request to

Figure 6 **CheckAvailability**

```
static public bool CheckAvailability(string uri) {
  bool retval = true;
  Binding binding = new NetTcpBinding(SecurityMode.None);
  EndPointServicesRef.EndPointServicesClient endpointsvc =
    new EndPointServicesRef.EndPointServicesClient(binding,
    new EndpointAddress(@"net.tcp://" + uri));
  endpointsvc.InnerChannel.OperationTimeout =
    new System.TimeSpan(0,0,0,0, 5000);

  try {
    retval = endpointsvc.IsAvailable();
  }
  catch (Exception ex) {
    // Todo: handle exception
    retval = false;
  }
  return retval;
}
```

Figure 7 **Demo Page Code**

```
protected void Page_Load(object sender, EventArgs e) {
  string UriString = EndpointManager.GetEndPoint();
  LastUri=UriString;

  Binding binding = new NetTcpBinding(SecurityMode.None);

  EndPointServicesRef.EndPointServicesClient endpointsvc =
    new EndPointServicesRef.EndPointServicesClient(binding,
    new EndpointAddress(@"net.tcp://" + UriString));
  lblMessage.Text = "WebInstacne ID: " +
    RoleEnvironment.CurrentRoleInstance.Id.ToString() +
    " is Calling Service @ " + UriString + " & IsAvailable = " +
    endpointsvc.IsAvailable().ToString();
  cmdDisable.Enabled=true;
}
```

balance, thus the endpoint also shows up differently. If I click the disable button, a small piece of code runs to set call DisableNode for the current endpoint:

```
protected void cmdDisable_Click(object sender, EventArgs e) {
  Binding binding = new NetTcpBinding(SecurityMode.None);
  EndPointServicesRef.EndPointServicesClient endpointsvc =
    new EndPointServicesRef.EndPointServicesClient(binding,
    new EndpointAddress(@"net.tcp://" + LastUri));
  endpointsvc.DisableNode();
}
```

The DisableNode method simply sets the Boolean and then sets up a timer to enable it back. The timer is set to be a bit longer than the expiry for the cached endpoint so as to make it easier to illustrate this in the test run:

```
public void DisableNode() {
  AvailabilityState.Enabled = false;
  AvailabilityState.Available = false;

  System.Timers.Timer invalidateTimer =
    new System.Timers.Timer(20000);
  invalidateTimer.Elapsed += (sender, e) => EnableNode();
  invalidateTimer.Start();
}
```

With the node disabled, the subsequent requests coming from different Web servers should all balance to the same worker endpoint.

## Beyond the Example

This is obviously a trivial example to illustrate the point, but I want to highlight some things to consider for an actual implementation. I also want to mention David's implementation to solve the problem because he addressed a domain of problems that I did not.

It was my intent for this example that the calling node would run the endpoint resolution code as part of the role startup process. It would cache the endpoint in a static member or in an actual cache refreshing based on cache expiry. However, it could be combined as part of the service implementation, allowing fine-grain control versus the unit being at the level of the IP and port combo. Depending on the actual problem being solved and the design of the service fabric, I might choose one style over the other.

To get this running in a production environment, here are some things to consider and possibly resolve:

- The intelligence for deciding the availability. This means not only the things that might be examined (CPU, disk, back-end connection state and so on), but also the thresholds that should be used to flip the bit between being available or not.
- Logic to handle the case that all return unavailable.
- Decisions about the quantum to cache the endpoint.
- Some additional methods in the EndpointManager to change settings, remove nodes from the pool and general runtime maintenance.
- All of the typical exception handling and diagnostics usually included in a service implementation.

I realize that those things probably go without saying, but I like to stick with a guideline of "No Guessing."

> The calling node would run the endpoint resolution code as part of the role startup process.

In quick summary of David's approach, he set up a matrix between Fault Domains and Upgrade Domains in an attempt to ensure that the caller availability matched the endpoint availability by preferring endpoints in the same domains. I think this is a great idea. Combining my implementation with his would ensure that your Web is serviced by a Worker Role following the same service level agreement if at all possible, but in the case that none are available, it would have the ability to balance to any other node.

## Final Thoughts

I hope the Windows Azure platform will evolve to allow load balancing for private endpoints as a point of configuration. Until then, if it's something you need (raw sockets will almost always want a level of protection by being internal), then a code solution will probably be the easiest way to go. By segmenting the endpoint resolution calls away from the actual service calls and making them part of the startup, it should keep the value-add code clean and separate from the foundation code. Thus, once a feature like this becomes available to configure, the services should continue to work while allowing you to disable the balancing code. ■

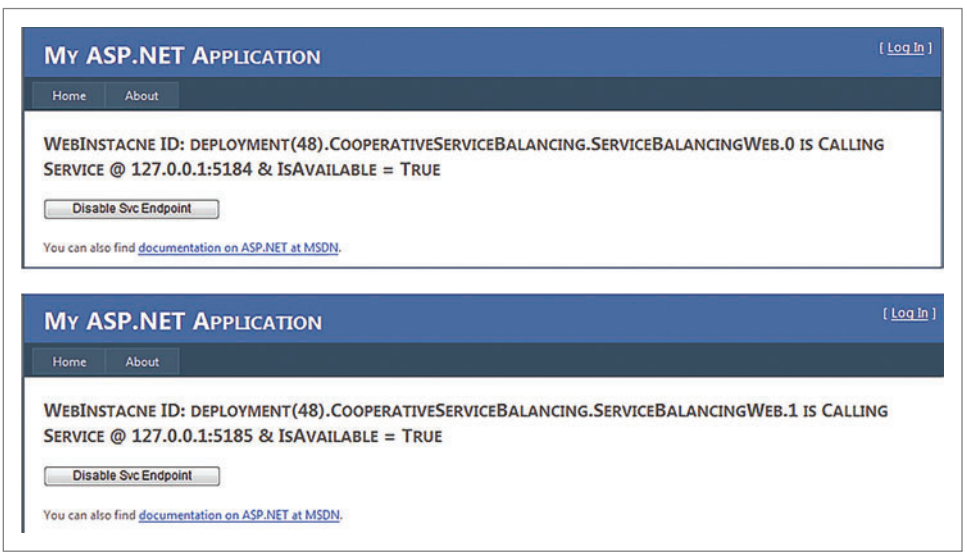



Figure 8 **Running the Demo**

**Joseph Fultz** *is an architect at the Microsoft Technology Center in Dallas, where he works with both enterprise customers and ISVs designing and prototyping software solutions to meet business and market demands. He's spoken at events such as Tech·Ed and similar internal training events.*

Your best source for software development tools!

programmer's paradise

## LEADTOOLS Document Imaging Suite SDK v17.0

**by LEAD Technologies**

- Libraries for C/C++, .NET, Silverlight, Windows Phone, WPF, WCF & WF
- High Performance OCR, ICR, MICR & OMR
- 1D & 2D Barcodes (Read/Write)
- Forms Recognition/Processing
- PDF, PDF/A and XPS
- Document Cleanup
- Advanced Compression (CCITT G3/G4, JBIG2, MRC, ABIC, ABC)
- High-Speed Scanning
- Print Capture and Document Writers

**programmers.com/LEAD**

Paradise # L05 03301A02

**$4,018.99**

## UltraEdit

*The #1 Best Selling Text Editor in the World*

**by IDM Computer Solutions**

UltraEdit is the world's standard in text editors. Millions use UltraEdit as the ideal text/hex/programmers editor on any platform — Windows, Mac, or Linux!

Features include syntax highlighting for nearly any programming language; powerful Find, Replace, Find in Files, and Replace in Files; FTP support, sort, column mode, hex, macros/scripting, large file handling (4+ GB), projects, templates, Unicode, and more.


Named User 1-24 Users
Paradise # I84 01201A01

**$59.95**

**programmers.com/idm**

## VMware vSphere Essentials Kit Bundle

vSphere Essentials provides an all-in-one solution for small offices to virtualize three physical servers for consolidating and managing applications to reduce hardware and operating costs with a low up-front investment. vSphere Essentials includes:

- VMware ESXi and VMware ESX (deployment-time choice)
- VMware vStorage VMFS
- Four-way virtual SMP
- VMware vCenter Server Agent
- VMware vStorage APIs/VCB
- VMware vCenter Update Manager
- VMware vCenter Server for Essentials


for 3 hosts
Paradise # V55 85101C02

**$446.99**

**programmers.com/vSphere**

## Embarcadero Delphi XE

*The Fastest Way to Build Native Windows Applications*

**by Embarcadero**

Embarcadero® Delphi® XE is the fastest way to deliver ultra-rich, ultra-fast Windows applications. Dramatically reduce coding time and create applications 5x faster with component-based development and a fully visual two-way RAD IDE. Speed development across multiple Windows and database platforms — for GUI desktop applications, interactive touch-screen, kiosk, and database-driven multi-tier, cloud, and Web applications.

NEW RELEASE!

Paradise # CGI 32401A01

**$1,977.99**

**programmers.com/embarcadero**

## Apple MacBook Pro MC700LL/A 13.3" LED Notebook

**by Apple**

MacBook Pro is machined from a single piece of aluminum, an engineering breakthrough that replaced many parts with just one. It's called the unibody. And the first time you pick up a MacBook Pro you'll notice the difference it makes. The entire enclosure is thinner and lighter than other notebooks. It looks polished and refined. And it feels strong and durable — perfect for life inside (and outside) your briefcase or backpack.

Paradise # ZHI BJ0000

**$1,199.99**

**programmers.com/apple**

## TX Text Control 16.0

*Word Processing Components*

TX Text Control is royalty-free, robust and powerful word processing software in reusable component form.

New Version Released!

- .NET WinForms and WPF rich text box for VB.NET and C#
- ActiveX for VB6, Delphi, VBScript/HTML, ASP
- File formats DOCX, DOC, RTF, HTML, XML, TXT
- PDF and PDF/A export, PDF text import
- Tables, headers & footers, text frames, bullets, structured numbered lists, multiple undo/redo, sections, merge fields, columns
- Ready-to-use toolbars and dialog boxes


Professional Edition
Paradise # T79 12101A01

**$1,109.99**

Download a demo today.

**programmers.com/textcontrol**

## Spread 5 for Windows Forms

**by GrapeCity PowerTools**

- World's best selling .NET Spreadsheet
- Import/export native Microsoft Excel files with full formatting
- Extremely flexible printing and export options including PDF
- Extensible formula support, including Microsoft Excel functions
- Hundreds of chart styles for enhanced data visualization
- Powerful user interface and flexible data connectivity
- WYSIWYG spreadsheet designers, quick-start wizard and chart designers
- Royalty-free licensing

**programmers.com/grapecity**

Upgrade
Paradise # F02 01101A01

**$936.99**

## Sony VAIO VPCF13DGX/B 16.4" Notebook

**by Sony**

There's something unique about every VAIO notebook. Born to perform, the latest VAIO F Series laptop is locked and loaded for all-out performance. Forged for gaming and built for quickness, the F embodies everything multimedia. Thanks to new Intel® Turbo Boost Technology, the Intel® Core™ i7 quad-core processor automatically adapts to your tasks, dialing up speeds and responding to your needs.

Paradise # ZHI DN0952

**$1,046.99**

**programmers.com/sony**

## HP EliteBook 8440p XT917UT 14" LED Notebook

**by Hewlett Packard**

For mobile professionals who need manageability, security, upgradeable wireless and enhanced system and graphics performance in a business-rugged notebook with a 14.0-inch diagonal display, the HP EliteBook 8440p Notebook PC features a brushed-aluminum platinum finish and business-rugged construction to protect your notebook on the go. A magnesium/aluminum display enclosure and magnesium alloy chassis provides increased durability.

Paradise # ZHI DQ4020

**$1,169.99**

**programmers.com/hp**

## StorageCraft ShadowProtect Desktop Edition 4.0

**by StorageCraft Technology Corp**

ShadowProtect Desktop provides fast and reliable disaster recovery, data protection and system migration to get desktops and laptops online as quickly as possible. ShadowProtect Desktop is an automated backup that works in the background. In the event of a disaster, its flexible recovery options allow you to complete granular recovery of files and folders or full bare metal recovery in a matter of minutes. Hardware Independent Restore (HIR) technology makes it quick and easy to recover to the same system, to dissimilar hardware or to and from virtual environments.

**programmers.com/storagecraft**

Minimum Quantity: 1
Paradise # SC5 02201E01

**$83.99**

## Lenovo ThinkPad T410 2516DCU 14.1" LED Notebook

**by Lenovo**

Rock-solid reliability. Blazing-fast performance. Clever manageability tools. We pack a lot of intelligent features into every ThinkPad laptop so you get more out. More productivity, more cost savings, more IT headache busting. That's why a ThinkPad investment isn't just smart. It's pure genius.

Paradise # ZHI DL7471

**$999.00**

**programmers.com/lenovo**

## Win an iPad!

NO PURCHASE NECESSARY.
Use offer code WEBTRW05 when you place your order online or with your Programmer's Paradise representative and you'll automatically be entered into a drawing to win an iPad Wi-Fi 32GB.

For official rules, or to complete the online entry form:
**programmers.com/tradewinds**

**866-719-1528**

**programmersparadise.com**

*Prices subject to change. Not responsible for typographical errors.*

# Writing a Debugging Tools for Windows Extension, Part 2: Output

Andrew Richards

In this second installment of my series about the Debugger API, I'm going to show you how you can enhance the output generated by your Debugger Engine (DbgEng) extension. You can fall into a number of traps when doing this, though. I hope to highlight all of the pitfalls for you.

Before reading on, you'll want to have read the previous installment to understand what a debugger extension is (and how I'm building and testing the examples in this article). You can read it at msdn.microsoft.com/magazine/gg650659.

This article discusses:

- Debugger Markup Language
- "Hello DML world"
- Markup
- Hyperlinks
- User preference
- Ability of the debugging client
- Controlled output
- Output control
- The mask parameter

Technologies discussed:

Debugging Tools for Windows, Debugger Markup Language

Code download available at:

code.msdn.microsoft.com/mag201105DebugOutput

## Debugger Markup Language

Debugger Markup Language (DML) is an HTML-inspired markup language. It supports emphasis via bolding/italics/underlining and navigation via hyperlinks. DML was added to the Debugger API in version 6.6.

The Windows SDK for Windows Vista first shipped version 6.6.7.5 of this API and supported x86, x64 and IA64. The Windows 7/.NET 3.5 SDK/WDK shipped the next release (version 6.11.1.404). The Windows 7/.NET 4 SDK/WDK ships the current release (version 6.12.2.633). The Windows 7/.NET 4 release vehicle is the only way to get the latest version of the Debugging Tools for Windows from Microsoft. There's no direct download of the x86, x64 or IA64 packages available. Note that these subsequent releases don't expand on the DML-related APIs defined in version 6.6. They do, however, have worthwhile fixes relating to DML support.

## 'Hello DML World'

As you can probably guess, the markup used by DML for emphasis is the same markup as used by HTML. To mark text as bold, use "…"; for italic, use "…"; and for underline use "<u>…</u>". **Figure 1** shows an example command that outputs "Hello DML World!" with these three types of markup.

To test the extension, I have a script called test_windbg.cmd in the Example04 folder of the code download accompanying this article. The script copies the extension to the C:\Debuggers_x86 folder. The script then starts WinDbg, loads the extension and launches a new instance of Notepad (as the debug target). If everything has gone according to plan, I can type "!hellodml" in

the debugger's command prompt and see a "Hello DML World!" response with bold, italic and underline markup:

```
0:000> !hellodml
Hello DML World!
```

I also have a script called test_ntsd.cmd that does the same steps, but loads the NTSD debugger. If I type "!hellodml" in this debugger's command prompt, I'll see the "Hello DML World!" response, but with no markup. The DML is converted to text because NTSD is a text-only debug client. All markup is stripped when DML is outputted to a text (only)-based client:

```
0:000> !hellodml
Hello DML World!
```

## Markup

Much like with HTML, you need to be careful to start and end any markup. **Figure 2** has a simple extension command (!echoasdml) that echoes the command argument as DML with markers before and after the DML output as text output.

This example sequence shows what happens if you don't close markup:

```
0:000> !echoasdml Hello World
[Start DML]
Hello World
[End DML]

0:000> !echoasdml <b>Hello World</b>
[Start DML]
Hello World
[End DML]

0:000> !echoasdml <b>Hello
[Start DML]
Hello
[End DML]

0:000> !echoasdml World</b>
[Start DML]
World
[End DML]
```

The "<b>Hello" command leaves bold enabled, causing all following extension and prompt output to be displayed as bold. This is regardless of whether the text was outputted in text or DML mode. As you'd expect, the subsequent closing of the bold markup reverts the state.

Another common issue is when the string has XML tags within it. The output could be either truncated, as happens in the first example here, or the XML tags could be stripped:

```
0:000> !echoasdml <xml
[Start DML]

0:000> !echoasdml <xml>Hello World</xml>
[Start DML]
Hello World
[End DML]
```

You handle this in the same way as you would with HTML: escape sequence the string before output. You can do this yourself or have the debugger do it for you. The four characters that need escaping are &, <, > and ". The equivalent escaped versions are: "&", "<", ">" and """.

**Figure 3** shows an example of escape sequencing.

The echoasdmlescape command allocates a new buffer that is six times the size of the original. This is enough space to handle an argument string with purely " characters. The function iterates over the argument string (which is always ANSI) and adds the appropriate text to the buffer. It then uses the escape-sequenced buffer with the %s formatter passed to the IDebugClient::ControlledOutput function. The !echoasdmlescape command echoes the argument without the string being interpreted as DML markup:

```
0:000> !echoasdmlescape <xml
[Start DML]
<xml
[End DML]

0:000> !echoasdmlescape <xml>Hello World</xml>
[Start DML]
<xml>Hello World</xml>
[End DML]
```

Note that with some strings, you may still not get the output you expect, given the input provided. These inconsistencies don't have anything to do with the escape sequencing (or DML); they're caused by the debugger's parser. The two cases of note are the " character (string content) and the ";" character (command termination):

```
0:000> !echoasdmlescape "Hello World"
[Start DML]
Hello World
[End DML]

0:000> !echoasdmlescape Hello World;
[Start DML]
Hello World
[End DML]

0:000> !echoasdmlescape "Hello World;"
[Start DML]
Hello World;
[End DML]
```

You don't have to go through this escape sequencing effort yourself, though. The debugger supports a special formatter for this case. Instead of generating the escape sequenced string and then using the %s formatter, you can just use the %Y{t} formatter on the original string.

Figure 1 **!hellodml Implementation**

```
HRESULT CALLBACK
hellodml(PDEBUG_CLIENT pDebugClient, PCSTR args)
{
  UNREFERENCED_PARAMETER(args);

  IDebugControl* pDebugControl;
  if (SUCCEEDED(pDebugClient->QueryInterface(__uuidof(IDebugControl),
    (void **)&pDebugControl)))
  {
    pDebugControl->ControlledOutput(
      DEBUG_OUTCTL_AMBIENT_DML, DEBUG_OUTPUT_NORMAL,
      "<b>Hello</b> <i>DML</i> <u>World!</u>\n");
    pDebugControl->Release();
  }
  return S_OK;
}
```

Figure 2 **!echoasdml Implementation**

```
HRESULT CALLBACK
echoasdml(PDEBUG_CLIENT pDebugClient, PCSTR args)
{
  IDebugControl* pDebugControl;
  if (SUCCEEDED(pDebugClient->QueryInterface(__uuidof(IDebugControl),
    (void **)&pDebugControl)))
  {
    pDebugControl->Output(DEBUG_OUTPUT_NORMAL, "[Start DML]\n");
    pDebugControl->ControlledOutput(
      DEBUG_OUTCTL_AMBIENT_DML,
      DEBUG_OUTPUT_NORMAL, "%s\n", args);
    pDebugControl->Output(DEBUG_OUTPUT_NORMAL, "[End DML]\n");
    pDebugControl->Release();
  }
  return S_OK;
}
```

Figure 3 **!echoasdmlescape Implementation**

```
HRESULT CALLBACK
echoasdmlescape(PDEBUG_CLIENT pDebugClient, PCSTR args)
{
  IDebugControl* pDebugControl;
  if (SUCCEEDED(pDebugClient->QueryInterface(__uuidof(IDebugControl),
    (void **)&pDebugControl)))
  {
    pDebugControl->Output(DEBUG_OUTPUT_NORMAL, "[Start DML]\n");
    if ((args != NULL) && (strlen(args) > 0))
    {
      char* szEscape = (char*)malloc(strlen(args) * 6);
      if (szEscape == NULL)
      {
        pDebugControl->Release();
        return E_OUTOFMEMORY;
      }
      size_t n=0; size_t e=0;
      for (; n<strlen(args); n++)
      {
        switch (args[n])
        {
          case '&':
                memcpy(&szEscape[e], "&amp;", 5);
                e+=5;
                break;
          case '<':
                memcpy(&szEscape[e], "&lt;", 4);
                e+=4;
                break;
          case '>':
                memcpy(&szEscape[e], "&gt;", 4);
                e+=4;
                break;
          case '"':
                memcpy(&szEscape[e], "&quot;", 6);
                e+=6;
                break;
          default:
                szEscape[e] = args[n];
                e+=1;
                break;
        }
      }
      szEscape[e++] = '\0';
      pDebugControl->ControlledOutput(DEBUG_OUTCTL_AMBIENT_DML,
        DEBUG_OUTPUT_NORMAL, "%s\n", szEscape);
      free(szEscape);
    }
    pDebugControl->Output(DEBUG_OUTPUT_NORMAL, "[End DML]\n");
    pDebugControl->Release();
  }
  return S_OK;
}
```

You can also avoid the escape sequencing effort if you use a memory formatter. The %ma, %mu, %msa and %msu formatters allow you to output a string directly from the target's address space; the debugging engine handles the reading of the string and the display, as shown in **Figure 4**.

In the second and third examples in **Figure 4**, the strings printed with %s are truncated or omitted due to the < and > characters, but the %ma and %Y{t} output is correct. The Test02 application is in **Figure 5**.

The !memorydml implementation is in **Figure 6**.

The test scripts (in the Example05 folder) have been changed to load a dump of the Test02 application instead of launching Notepad so that you have a string to output.

So, the easiest way to implement the display of strings from the target's address space is to just use %ma and so on. If you need to manipulate a string that has been read prior to display, or have just made a string of your own, then apply the escape sequencing via %Y{t}. If you need to pass the string as the format string, then apply the escape sequencing yourself. Alternatively, split the output into multiple IDebugControl::ControlledOutput calls and just use the DML output control (DEBUG_OUTCTL_AMBIENT_DML) on the DML part of the content, outputting the rest as TEXT (DEBUG_OUTCTL_AMBIENT_TEXT) with no escape sequencing.

The other constraint in this area is the length limit of IDebugClient::ControlledOutput and IDebugClient::Output; they can only output about 16,000 characters at a time. I've found that I regularly hit this limit with ControlledOutput when doing DML output. The markup and the escape sequencing can easily bloat a string past 16,000 characters while still looking relatively small in the output window.

> When you're building large strings and are doing the escape sequencing yourself, you'll need to make sure that you cut up the string at appropriate points.

When you're building large strings and are doing the escape sequencing yourself, you'll need to make sure that you cut up the string at appropriate points. Don't cut them up within the DML markup or within an escape sequence. Otherwise, they won't be interpreted correctly.

## Hyperlinks

There are two ways to achieve a hyperlink in the debugger: you can use <link> or <exec> markup. In both markups, the enclosed text is displayed with underscore and hypertext coloring (usually blue). The command that's executed in both cases is the "cmd" member. It's similar to the "href" member of the <a> markup in HTML:

```
<link cmd="dps @$csp @$csp+0x80">Stack</link>
<exec cmd="dps @$csp @$csp+0x80">Stack</exec>
```

Figure 4 **Reading the String and Display from a Memory Formatter**

```
0:000> !memorydml test02!g_ptr1
[Start DML]
Error (  %ma): File not found
Error (%Y{t}): File not found
Error (   %s): File not found
[End DML]

0:000> !memorydml test02!g_ptr2
[Start DML]
Error (  %ma): Value is < 0
Error (%Y{t}): Value is < 0
Error (   %s): Value is [End DML]

0:000> !memorydml test02!g_ptr3
[Start DML]
Error (  %ma): Missing <xml> element
Error (%Y{t}): Missing <xml> element
Error (   %s): Missing  element
[End DML]
```

# THESE GUYS STAND ON THEIR OWN.

That's because we focus on the most important controls, not dozens of generic, bundled ones.


XCEED **DataGrid** for WPF

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

XCEED **DataGrid** for Silverlight

The only Silverlight datagrid on the market with fast remote data retrieval and a completely fluid UI!

XCEED **Ultimate ListBox** for Silverlight

The same data virtualization and smooth-scrolling as our Silverlight datagrid, packed in the streamlined format of a listbox.

Try them live at **xceed.com**

XCEED
MULTI-TALENTED COMPONENTS

Figure 5 **Test02 Implementation**

```
// Test02.cpp : Defines the entry point for the console application.
//

#include <windows.h>

void* g_ptr1;
void* g_ptr2;
void* g_ptr3;

int main(int argc, char* argv[])
{
  g_ptr1 = "File not found";
  g_ptr2 = "Value is < 0";
  g_ptr3 = "Missing <xml> element";
  Sleep(10000);
  return 0;
}
```

The difference between <link> and <exec> can be hard to see. Actually, it took me quite a bit of research to get to the bottom of it. The difference is only observable in the Command Browser (Ctrl+N) window, not the Output window (Alt+1). In both types of window, the output of the <link> or <exec> link is displayed in the associated output window. The difference is what happens to the command prompt of each window.

In the Output window, the content of the command prompt doesn't change. If there's some unexecuted text there, it remains unchanged.

In the Command Browser window, when a <link> is invoked, the command is added to the list of commands and the command prompt is set to the command. But when an <exec> is invoked, the command isn't added to the list and the command prompt isn't changed. By not changing the command history, it's possible to make a sequence of hyperlinks that can guide the user through a decision process. The most common example of this is the displaying of help. The navigation around help is well-suited to hyperlinking, yet it's desirable not to log the navigation.

## User Preference

So how do you know if the user even wants to see DML? In a few scenarios, data issues can occur when the output is converted to text, be it by the action of saving the text to a log file (.logopen) or copying and pasting from the output window. Data can be lost due to DML abbreviation, or the data may be superfluous due to the inability to do navigation via the text version of the output.

Similarly, if it's onerous to generate the DML output, this effort should be avoided if it's known that this output will be content-converted. Lengthy operations usually involve memory scans and symbol resolution.

Equally, users might just prefer not to have DML in their output.

In this abbreviation example that's <link>-based, you'd only get "Object found at 0x0C876C32" outputted and miss the important piece of information (the data type of the address):

```
Object found at <link cmd="dt login!CSession 0x0C876C32">0x0C876C32</link>
```

The correct way of handling this is to have a condition that avoids the abbreviation when DML isn't enabled. Here's an example of how you could fix this:

```
if (DML)
  Object found at <link cmd="dt login!CSession 0x0C876C32">0x0C876C32</link>
else
  Object found at 0x0C876C32 (login!CSession)
```

The .prefer_dml setting is the closest you can get to a user preference (so you can make this conditional decision about abbreviated or superfluous output). The setting is used to control whether the debugger runs DML-enhanced versions of the built-in commands and operations by default. Although it isn't explicitly meant to specify whether DML should be used globally (within extensions), it's a good substitute.

The only downside of this preference is that the preference defaults to off, and most debugging engineers don't know that the .prefer_dml command exists.

Note that an extension, not the debugging engine, has to have code to detect the ".prefer_dml" preference or the "ability" (I'll explain "ability" shortly). The debugging engine won't strip the DML output for you based on this setting; if the debugger is DML-capable, it will output in DML regardless.

To get the current ".prefer_dml" preference, you need to do a QueryInterface on the passed IDebugClient interface for the

Figure 6 **!memorydml Implementation**

```
HRESULT CALLBACK
memorydml(PDEBUG_CLIENT pDebugClient, PCSTR args)
{
  IDebugDataSpaces* pDebugDataSpaces;
  if (SUCCEEDED(pDebugClient->QueryInterface(__uuidof(IDebugDataSpaces),
    (void **)&pDebugDataSpaces)))
  {
    IDebugSymbols* pDebugSymbols;
    if (SUCCEEDED(pDebugClient->QueryInterface(__uuidof(IDebugSymbols),
      (void **)&pDebugSymbols)))
    {
      IDebugControl* pDebugControl;
      if (SUCCEEDED(pDebugClient->QueryInterface(__uuidof(IDebugControl),
        (void **)&pDebugControl)))
      {
        // Resolve the symbol
        ULONG64 ulAddress = 0;
        if ((args != NULL) && (strlen(args) > 0) &&
          SUCCEEDED(pDebugSymbols->GetOffsetByName(args, &ulAddress)))
        { // Read the value of the pointer from the target address space
          ULONG64 ulPtr = 0;
          if (SUCCEEDED(pDebugDataSpaces->
            ReadPointersVirtual(1, ulAddress, &ulPtr)))
          {
            char szBuffer[256];
            ULONG ulBytesRead = 0;
            if (SUCCEEDED(pDebugDataSpaces->ReadVirtual(
              ulPtr, szBuffer, 255, &ulBytesRead)))
            {
              szBuffer[ulBytesRead] = '\0';

              // Output the value via %ma and %s
              pDebugControl->ControlledOutput(DEBUG_OUTCTL_AMBIENT_TEXT,
                DEBUG_OUTPUT_NORMAL, "[Start DML]\n");
              pDebugControl->ControlledOutput(DEBUG_OUTCTL_AMBIENT_DML,
                DEBUG_OUTPUT_ERROR, "<b>Error</b> (  %%ma): %ma\n", ulPtr);
              pDebugControl->ControlledOutput(DEBUG_OUTCTL_AMBIENT_DML,
                DEBUG_OUTPUT_ERROR, "<b>Error</b> (%%Y{t}): %Y{t}\n", szBuffer);
              pDebugControl->ControlledOutput(DEBUG_OUTCTL_AMBIENT_DML,
                DEBUG_OUTPUT_ERROR, "<b>Error</b> (  %%s): %s\n", szBuffer);
              pDebugControl->ControlledOutput(DEBUG_OUTCTL_AMBIENT_TEXT,
                DEBUG_OUTPUT_NORMAL, "[End DML]\n");
            }
          }
        }
        pDebugControl->Release();
      }
      pDebugSymbols->Release();
    }
    pDebugDataSpaces->Release();
  }
  return S_OK;
}
```

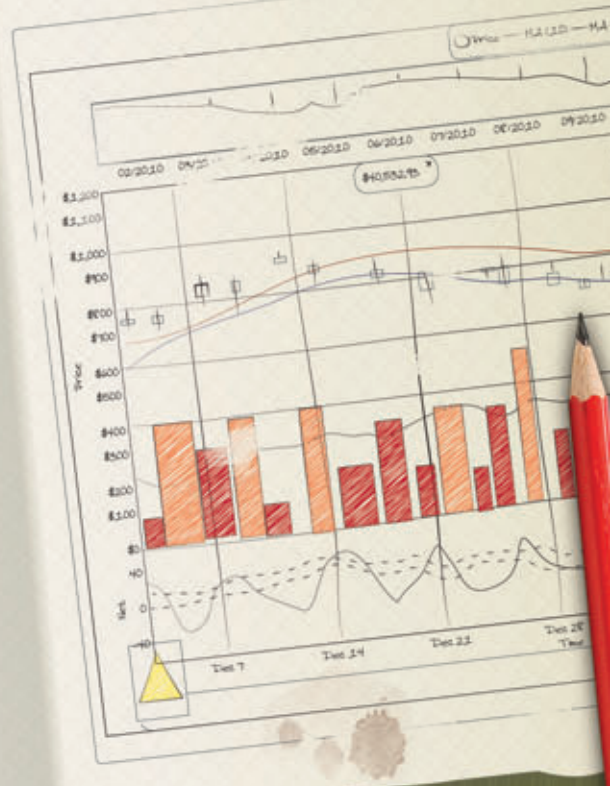
# Your Guide to .NET Charts

## Ahead of Schedule and Under Budget

With the right tools you can turn your long development list of requirements into a finished solution, from conception to completion. Developers, just like you, continue to turn to ComponentOne Charts™ for their enterprise solutions. Get rock solid, advanced charts for all .NET platforms; download ComponentOne Studio® Enterprise today.

Requirements:

- ✓ Financial & scientific charts
- ✓ Stacked charts
- ✓ Interactivity - zoom, animation, drag-and-drop
- ✓ Tooltips and markers
- ✓ Trend lines
- ✓ ~~Fast~~ Really fast
- ✓ Live updates
- ✓ 2D & 3D
- ✓ Multiple platforms - Silverlight, WPF, ASP.NET and WinForms




ComponentOne® Studio® Enterprise

Download your FREE Trial @ c1.ms/fastcharts

ComponentOne

Figure 7 **PreferDML Implementation**

```
BOOL PreferDML(PDEBUG_CLIENT pDebugClient)
{
  BOOL bPreferDML = FALSE;
  IDebugControl* pDebugControl;
  if (SUCCEEDED(pDebugClient->QueryInterface(__uuidof(IDebugControl),
    (void **)& pDebugControl)))
  {
    ULONG ulOptions = 0;
    if (SUCCEEDED(pDebugControl->GetEngineOptions(&ulOptions)))
    {
      bPreferDML = (ulOptions & DEBUG_ENGOPT_PREFER_DML);
    }
  pDebugControl->Release();
  }
  return bPreferDML;
}
```

IDebugControl interface. You then use the GetEngineOptions function to get the current DEBUG_ENGOPT_XXX bitmask. If the DEBUG_ENGOPT_PREFER_DML bit is set, .prefer_dml is enabled. **Figure 7** has an example implementation of a user preference function.

You may be thinking that you don't want to call the GetEngineOptions function in every command to determine the preference. Can't we be notified of the change? After all, it probably won't change very often. Yes, you can be more optimal, but there's a catch.

What you can do is register an IDebugEventCallbacks implementation via IDebugClient::SetEventCallbacks. In the implementation, you register an interest in the DEBUG_EVENT_CHANGE_ENGINE_STATE notification. When IDebugControl::SetEngineOptions is called, the debugger invokes IDebugEventCallbacks::ChangeEngineState with the DEBUG_CES_ENGINE_OPTIONS bit set in the Flags parameter. The Argument parameter contains a DEBUG_ENGOPT_XXX bitmask like GetEngineOptions returns.

The catch is that only one event callback can be registered at any one time for an IDebugClient object. If two (or more) extensions want to register for event callbacks (which includes more important notifications such as module load/unload, thread start/stop, process start/stop and exceptions), someone is going to miss out. And if you modify the passed IDebugClient object, that someone will be the debugger!

If you want to implement the IDebugEventCallbacks callback, you need to make your own IDebugClient object via IDebugClient::CreateClient. You then associate your callback with this (new) IDebugClient object and become responsible for the lifetime of the IDebugClient.

For simplicity's sake, you're better off calling GetEngineOptions each time you need to determine the DEBUG_ENGOPT_PREFER_DML value. As mentioned before, you should call QueryInterface on the passed IDebugClient interface for the IDebugControl interface, and then call GetEngineOptions to be sure that you have the current (and correct) preference.

## Ability of the Debugging Client

So how do you know if the debugger even supports DML?

If the debugger doesn't support DML, data can be lost, superfluous or the effort can be onerous, much like the user preference. As mentioned, NTSD is a text-only debugger, and if DML is outputted to it, the debugging engine does content conversion to remove the DML from the output.

To get the debugging client's ability, you need to do a QueryInterface on the passed IDebugClient interface for the IDebugAdvanced2 interface. You then use the Request function with the DEBUG_REQUEST_CURRENT_OUTPUT_CALLBACKS_ARE_DML_AWARE request type. The HRESULT contains S_OK when at least one Output Callback is DML-aware, otherwise it returns S_FALSE. To reiterate, the flag doesn't mean that all callbacks are aware; it means that *at least one* is.

In seemingly text-only environments (such as NTSD) you can still get into the conditional output issues. If an extension registers an output callback that's DML-aware (by returning DEBUG_OUTCBI_DML or DEBUG_OUTCBI_ANY_FORMAT from IDebugOutputCallbacks2::GetInterestMask) within NTSD, it will cause the Request function to return S_OK. Luckily, these extensions are quite rare. If they do exist, they should be checking the state of DEBUG_REQUEST_CURRENT_OUTPUT_CALLBACKS_ARE_DML_AWARE and setting their ability accordingly (prior to their proclamation of their DML ability). Check out the next installment of this series for more information about DML-aware callbacks.

**Figure 8** has an example implementation of an ability function.

> If the debugger doesn't support DML, data can be lost, superfluous or the effort can be onerous, much like the user preference.

Note that the DEBUG_REQUEST_CURRENT_OUTPUT_CALLBACKS_ARE_DML_AWARE request type and IDebugOutputCallbacks2 interface aren't documented in the MSDN Library yet.

Keeping the potential shortfalls in mind, the best way of handling user preference and client ability is:

```
if (PreferDML(IDebugClient) && AbilityDML(IDebugClient))
  Object found at <link cmd="dt login!CSession 0x0C876C32">0x0C876C32</link>
else
  Object found at 0x0C876C32 (login!CSession)
```

The !ifdml implementation (in **Figure 9**) shows the PreferDML and AbilityDML functions in action so that conditional DML

Figure 8 **AbilityDML Implementation**

```
BOOL AbilityDML(PDEBUG_CLIENT pDebugClient)
{
  BOOL bAbilityDML = FALSE;
  IDebugAdvanced2* pDebugAdvanced2;
  if (SUCCEEDED(pDebugClient->QueryInterface(__uuidof(IDebugAdvanced2),
    (void **)& pDebugAdvanced2)))
  {
    HRESULT hr = 0;
    if (SUCCEEDED(hr = pDebugAdvanced2->Request(
      DEBUG_REQUEST_CURRENT_OUTPUT_CALLBACKS_ARE_DML_AWARE,
      NULL, 0, NULL, 0, NULL)))
    {
      if (hr == S_OK) bAbilityDML = TRUE;
    }
    pDebugAdvanced2->Release();
  }
  return bAbilityDML;
}
```

# DESIGN

**Design Applications That Help Run the Business**


Our xamMap™ control in Silverlight and WPF lets you map out any geospatial data like this airplane seating app to manage your business. Come to infragistics.com to try it today!

NetAdvantage® ULTIMATE

for ASP.NET, Windows Forms, WPF, Silverlight,
WPF Data Visualization, Silverlight Data Visualization

Infragistics®

Infragistics Sales 800 231 8588
Infragistics Europe Sales +44 (0) 800 298 9055
Infragistics India +91 80 4151 8042
@infragistics

Figure 9 **!ifdml Implementation**

```
HRESULT CALLBACK
ifdml(PDEBUG_CLIENT pDebugClient, PCSTR args)
{
  UNREFERENCED_PARAMETER(args);

  PDEBUG_CONTROL pDebugControl;
  if (SUCCEEDED(pDebugClient->QueryInterface(__uuidof(IDebugControl),
    (void **)&pDebugControl)))
  {
    // A condition is usually not required;
    // Rely on content conversion when there isn't
    // any abbreviation or superfluous content
    if (PreferDML(pDebugClient) && AbilityDML(pDebugClient))
    {
      pDebugControl->ControlledOutput(DEBUG_OUTCTL_AMBIENT_DML,
        DEBUG_OUTPUT_NORMAL, "<b>Hello</b> <i>DML</i> <u>World!</u>\n");
    }
    else
    {
      pDebugControl->ControlledOutput(
        DEBUG_OUTCTL_AMBIENT_TEXT, DEBUG_OUTPUT_NORMAL,
        "Hello TEXT World!\n");
    }
    pDebugControl->Release();
  }
  return S_OK;
}
```

output is generated. Note that in the vast majority of cases, there's no need to have a conditional statement such as this; you can safely rely on the debugger engine content conversion.

Using the test_windbg.cmd test script to load WinDbg, the output of !ifdml is:

```
0:000> .prefer_dml 0
DML versions of commands off by default
0:000> !ifdml
Hello TEXT World!

0:000> .prefer_dml 1
DML versions of commands on by default
0:000> !ifdml
Hello DML World!
```

Using the test_ntsd.cmd test script to load NTSD, the output of !ifdml is:

```
0:000> .prefer_dml 0
DML versions of commands off by default
0:000> !ifdml
Hello TEXT World!

0:000> .prefer_dml 1
DML versions of commands on by default
0:000> !ifdml
Hello TEXT World!
```

## Controlled Output

To output DML, you need to use the IDebugControl::ControlledOutput function:

```
HRESULT ControlledOutput(
  [in]  ULONG OutputControl,
  [in]  ULONG Mask,
  [in]  PCSTR Format,
        ...
);
```

The difference between ControlledOutput and Output is the OutputControl parameter. This parameter is based on the DEBUG_OUTCTL_XXX constants. There are two parts to this parameter: the lower bits represent the scope of the output, and the higher bits represent the options. It's a higher bit that enables DML.

One—and only one—of the DEBUG_OUTCTL_XXX scope-based constants must be used for the lower bits. The value directs where the output is to go. This can be to all debugger clients (DEBUG_OUTCTL_ALL_CLIENTS), just the IDebugClient associated with the IDebugControl interface (DEBUG_OUTCTL_THIS_CLIENT), all other clients (DEBUG_OUTCTL_ALL_OTHER_CLIENTS), nowhere at all (DEBUG_OUTCTL_IGNORE) or just the log file (DEBUG_OUTCTL_LOG_ONLY).

The higher bits are a bit mask and are also defined in the DEBUG_OUTCTL_XXX constants. There are constants to specify text- or DML-based output (DEBUG_OUTCTL_DML), if the output isn't logged (DEBUG_OUTCTL_NOT_LOGGED) and whether a client's output mask is honored (DEBUG_OUTCTL_OVERRIDE_MASK).

## Output Control

In all the examples, I've set the ControlledOutput parameter to DEBUG_OUTCTL_AMBIENT_DML. Reading the documentation on MSDN, you might say that I could've also used DEBUG_OUTCTL_ALL_CLIENTS | DEBUG_OUTCTL_DML. However, this wouldn't honor the IDebugControl output control preference.

If the extension's command was invoked by IDebugControl::Execute, the OutputControl parameter of the Execute call should be used for any related output. IDebugControl::Output does this inherently, but when using IDebugControl::ControlledOutput, the responsibility of knowing the OutputControl value is the caller's. The issue is

Figure 10 **!maskdml Implementation**

```
HRESULT CALLBACK
maskdml(PDEBUG_CLIENT pDebugClient, PCSTR args)
{
  UNREFERENCED_PARAMETER(args);

  PDEBUG_CONTROL pDebugControl;
  if (SUCCEEDED(pDebugClient->QueryInterface(__uuidof(IDebugControl),
    (void **)&pDebugControl)))
  {
    pDebugControl->ControlledOutput(
      DEBUG_OUTCTL_AMBIENT_DML, DEBUG_OUTPUT_NORMAL,
      "<b>DEBUG_OUTPUT_NORMAL</b> - Normal output.\n");
    pDebugControl->ControlledOutput(DEBUG_OUTCTL_AMBIENT_DML,
      DEBUG_OUTPUT_ERROR, "<b>DEBUG_OUTPUT_ERROR</b> - Error output.\n");
    pDebugControl->ControlledOutput(DEBUG_OUTCTL_AMBIENT_DML,
      DEBUG_OUTPUT_WARNING, "<b>DEBUG_OUTPUT_WARNING</b> - Warnings.\n");
    pDebugControl->ControlledOutput(DEBUG_OUTCTL_AMBIENT_DML,
      DEBUG_OUTPUT_VERBOSE, "<b>DEBUG_OUTPUT_VERBOSE</b>
      - Additional output.\n");
    pDebugControl->ControlledOutput(
      DEBUG_OUTCTL_AMBIENT_DML, DEBUG_OUTPUT_PROMPT,
      "<b>DEBUG_OUTPUT_PROMPT</b> - Prompt output.\n");
    pDebugControl->ControlledOutput(DEBUG_OUTCTL_AMBIENT_DML,
      DEBUG_OUTPUT_PROMPT_REGISTERS, "<b>DEBUG_OUTPUT_PROMPT_REGISTERS</b>
      - Register dump before prompt.\n");
    pDebugControl->ControlledOutput(DEBUG_OUTCTL_AMBIENT_DML,
      DEBUG_OUTPUT_EXTENSION_WARNING,
      "<b>DEBUG_OUTPUT_EXTENSION_WARNING</b>
      - Warnings specific to extension operation.\n");
    pDebugControl->ControlledOutput(DEBUG_OUTCTL_AMBIENT_DML,
      DEBUG_OUTPUT_DEBUGGEE, "<b>DEBUG_OUTPUT_DEBUGGEE</b>
      - Debug output from the target (for example, OutputDebugString or
      DbgPrint).\n");
    pDebugControl->ControlledOutput(DEBUG_OUTCTL_AMBIENT_DML,
      DEBUG_OUTPUT_DEBUGGEE_PROMPT, "<b>DEBUG_OUTPUT_DEBUGGEE_PROMPT</b>
      - Debug input expected by the target (for example, DbgPrompt).\n");
    pDebugControl->ControlledOutput(DEBUG_OUTCTL_AMBIENT_DML,
      DEBUG_OUTPUT_SYMBOLS, "<b>DEBUG_OUTPUT_SYMBOLS</b>
      - Symbol messages (for example, !sym noisy).\n");
    pDebugControl->Release();
  }
  return S_OK;
}
```

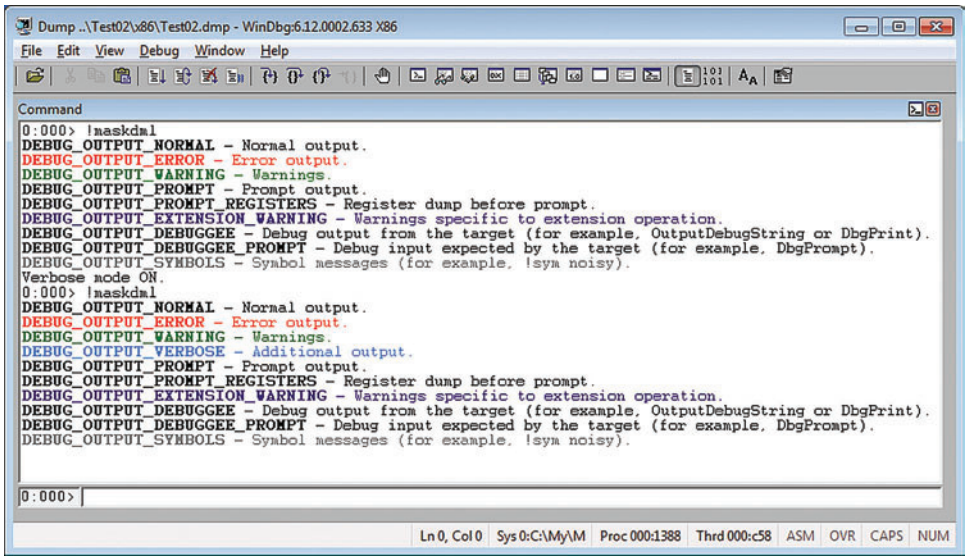



Figure 11 **!maskdml with Color Scheme**

that there's no way to actually retrieve the current output control value from the IDebugControl interface (or any other interface). All is not lost, though; there are special DEBUG_OUTCTL_XXX "ambient" constants to handle the toggling of the DEBUG_OUTCTL_DML bit. When you use an ambient constant, the current output control is honored and just the DEBUG_OUTCTL_DML bit is set accordingly.

Instead of passing one of the lower constants with the higher DEBUG_OUTCTL_DML constant, you simply pass DEBUG_OUTCTL_AMBIENT_DML to enable DML output, or DEBUG_OUTCTL_AMBIENT_TEXT to disable DML output:

```
pDebugControl->ControlledOutput(DEBUG_OUTCTL_AMBIENT_DML, ...);
```

## The Mask Parameter

Another parameter that I've been setting in the examples is the Mask parameter. You should set the Mask to an appropriate DEBUG_OUTPUT_XXX constant based on the text being outputted. Note that the Mask parameter is based on the DEBUG_OUTPUT_XXX constants; this isn't to be confused with the DEBUG_OUTCTL_XXX constants.

The most common values that you'd use are DEBUG_OUTPUT_NORMAL for normal (general) output, DEBUG_OUTPUT_WARNING for warning output and DEBUG_OUTPUT_ERROR for error output. You should use DEBUG_OUTPUT_EXTENSION_WARNING when your extension has an issue.

The DEBUG_OUTPUT_XXX output flags are similar to stdout and stderr used with console output. Each output flag is an individual output channel. It's up to the receivers (callbacks) to decide which of these channels will be listened to, how they're to be combined (if it all) and how they're to be displayed. For example, WinDbg by default displays all output flags except the DEBUG_OUTPUT_VERBOSE output flag in the Output window. You can toggle this behavior via View | Verbose Output (Ctrl+Alt+V).

The !maskdml implementation (see **Figure 10**) outputs a description styled with the associated output flag.

If you toggle Verbose Output after the command has run, the omitted DEBUG_OUTPUT_VERBOSE output won't be shown; the output is lost.

WinDbg supports colorization of each output flag. In the View | Options dialog, you can specify a foreground and background color for each output flag. The color settings are saved in the workspace. To set them globally, launch WinDbg, delete all the workspaces, set the colors (and any other setting you would like) and save the workspace. I like to set the foreground color of Error to red, Warning to green, Verbose to blue, Extension Warning to purple and Symbols to gray. The default workspace will be the template for all future debugging sessions.

**Figure 11** shows the output of !maskdml without (top) and with (bottom) verbose enabled.

## Break!

It's easy to enhance any extension with DML. And, with a little bit of infrastructure code, it's also easy to honor the user preference. It's definitely worth spending some extra time to generate output correctly. In particular, strive to always provide both text- and DML-based output when the output is abbreviated or superfluous, and direct this output appropriately.

In the next installment about the Debugger Engine API, I'm going to delve deeper into the relationship a debugger extension can have with the debugger. I'll give you an overview of the architecture of debugger clients and debugger callbacks. In doing so, we'll get into the nitty-gritty of the DEBUG_OUTPUT_XXX and DEBUG_OUTCTL_XXX constants.

I'll use this foundation to implement an encapsulation of the Son of Strike, or SOS, debugger extension. I'll enhance the SOS output with DML and demonstrate how you can leverage the built-in debugger commands and other extensions to retrieve information required by your extensions.

If you're just interested in debugging and want to learn more, you should check out the "Advanced Windows Debugging and Troubleshooting" (NTDebugging) blog at blogs.msdn.com/b/ntdebugging—there are lots of training and case study articles to read.

Microsoft is always looking for talented debugging engineers. If you're interested in joining the team, the job title to search for is "Escalation Engineer" at Microsoft Careers (careers.microsoft.com). ■

**ANDREW RICHARDS** *is a Microsoft senior escalation engineer for Exchange Server. He has a passion for support tools and is continually creating debugger extensions and applications that simplify the job of support engineers.*

**THANKS** to the following technical expert for reviewing this article: *Drew Bliss*

# EXPERIENCE

**Beautiful Data Visualizations That Bring Your Data to Life**



Use our Motion Framework™ to see your data over time and give your users new insight into their data. Visit infragistics.com/motion to try it today!

## NetAdvantage® ULTIMATE

for ASP.NET, Windows Forms, WPF, Silverlight,
WPF Data Visualization, Silverlight Data Visualization

Infragistics®

Infragistics Sales 800 231 8588
Infragistics Europe Sales +44 (0) 800 298 9055
Infragistics India +91 80 4151 8042
@infragistics

# Moving Your Web App from WebMatrix to ASP.NET MVC 3

Brandon Satrom and Clark Sell

This past January, Microsoft introduced a new programming model for Web development with the Microsoft .NET Framework called ASP.NET Web Pages. Currently supported by default in WebMatrix (web.ms/WebMatrix), Web Pages is a page-centric programming model that behaves much like PHP, where each page contains its own business logic, data access and dynamic content for rendering HTML to the browser.

There are a variety of reasons to build a Web site in WebMatrix. But what if you know you want to move to Visual Studio at some point in the future? If ASP.NET MVC 3 is your end state, will you need to re-develop the site when the time for migration arises? If you're afraid of boxing yourself in with WebMatrix and Web Pages, fear not.

Web Pages—as a part of the core ASP.NET framework—was built with flexibility in mind. And while there are no technical limitations to Web Pages that would ever force you to move to ASP.NET MVC, there may be times where it makes sense for your team, product or company to do so.

In this article, we'll discuss some of the reasons why you might choose to migrate (as well as some of the reasons not to). We'll also discuss strategies for moving your Web Pages site to ASP.NET MVC if you choose to do so. We'll cover how to move from Pages to Views, how to handle business logic and helper code, and how to introduce Models in your application. Finally, we'll discuss and show how to preserve existing site URLs through routing, and supporting permanent redirects when necessary.

This article discusses:

- Moving from Pages to Views
- Introducing Controllers
- Migrating data access
- Preserving URL routing

Technologies discussed:

ASP.NET MVC 3, WebMatrix

Code download available at:

code.msdn.microsoft.com/mag201105ASPNET

## When to Migrate?

Before we dive into reasons to migrate from Web Pages to ASP.NET MVC, let's discuss a few reasons not to make such a move. For starters, you shouldn't move your site from Web Pages to ASP.NET MVC because you're afraid your application won't scale. Because Web Pages is built on top of ASP.NET, it offers many of the same performance characteristics of applications written for ASP.NET MVC or Web Forms. Obviously, each application type has a slightly different execution model, but there's nothing about ASP.NET MVC out of the box that makes it inherently more or less scalable than a Web Pages application. Scale and performance is just as much about the design decisions you make as you build your site, as it is about the underlying framework you choose.

It's also not a good idea to move your site from Web Pages to ASP.NET MVC because it's cool, sexy and you hear everyone is doing it, or just because you want to work with your site in Visual Studio—you can do that with Web Pages sites already. ASP.NET

MVC is a choice in Web application architecture, not a magic wand that makes your site instantly better. Migration from Web Pages to ASP.NET MVC is a choice, not a necessity. And while Microsoft has done a fantastic job of making this migration possible and painless, it will still cost you time, resources and money, as any migration would. For these reasons, it's important to make certain that you're migrating to ASP.NET MVC for the right reasons.

One valid reason might be because unit testing is important to you and your team. Because the Web Pages model is page-centric, it's not possible to use existing unit-testing tools with Web Pages sites. Web UI testing—using tools like WatiN or Selenium—is still possible, but code-level unit testing with tools like NUnit or MsTest is not. If your site has grown in complexity and unit testing is important to you, a migration to ASP.NET MVC makes sense.

If you prefer to unit test your code, chances are you also prefer some measure of separation of concerns in your applications. And while it's possible to create clean, separated code using helpers and code files in Web Pages, the model doesn't lend itself to this separation as naturally as ASP.NET MVC does. If separation of concerns is important to you, and you want an application architecture that encourages such separation, migration is a valid choice.

Beyond these two, other reasons for migration might come into play depending on the context of your site or organization. If you have a growing team and a site that's increasing in complexity and requires richer business functionality, you'd be wise to migrate. Migration may also be necessary in order to make better use of a richer development ecosystem for things such as source control, load testing and so on.

## Preparing for Migration

For this article, we're going to take the Photo Gallery template site that ships with WebMatrix and migrate it to ASP.NET MVC. The core of the migration process is moving from Pages to Models, Controllers and Views, but we need to perform a bit of prep work before we can get there.

> If you're afraid of boxing yourself in with WebMatrix and Web Pages, fear not.

Because we're detailing steps involved in a manual migration in this short article, we won't be able to give every step equal attention. Our goal is to address the major steps, and at least mention the minor considerations. We've also chosen to omit items that aren't related to the core of the overall conversion, such as data-access best practices, potential assembly structure, dependency injection and the like. These items are important, but many of them will come down to development culture and personal preference, and can all be addressed in your post-migration refactorings.

It's also important to note that we're not making use of the Open in Visual Studio feature in WebMatrix for this article, which will open your current site as a Web Site project in Visual Studio. You're welcome to use this option, even if you choose not to migrate to ASP.NET MVC, but we prefer to use the Visual Studio Web Application project type for ASP.NET MVC, starting with an empty site and migrate things over manually.

Figure 1 **Business Logic Contained in Login.cshtml**

```
Page.Title = "Login";
if (IsPost) {
  var email = Request["email"];
  if (email.IsEmpty()) {
    ModelState.AddError(
      "email", "You must specify an email address.");
  }
  var password = Request["password"];
  if (password.IsEmpty()) {
    ModelState.AddError(
      "password", "You must specify a password.");
  }

  if (ModelState.IsValid) {
    var rememberMe = Request["rememberMe"].AsBool();
    if (WebSecurity.Login(email, password, rememberMe)) {
      string returnUrl = Request["returnUrl"];
      if (!returnUrl.IsEmpty()) {
        Context.RedirectLocal(returnUrl);
      } else{
        Response.Redirect("~/");
      }
    } else {
      ModelState.AddFormError(
        "The email or password provided is incorrect.");
    }
  }
}
```

As such, we'll begin the migration by selecting File | New | Project and selecting an ASP.NET MVC 3 Web Application with the empty template, using Razor as the default view engine.

Once you have your target application set up, you'll need to perform some initial migration work. Here's a rundown of the initial steps:

1. Add any packages you're using in your Web Pages site into your ASP.NET MVC site via NuGet (nuget.org).
2. Add references to System.Web.Helpers, WebMatrix.Data and WebMatrix.WebData. Set each to Copy Local = true in the properties pane.
3. Move the contents of _AppStart.cshtml to the Application_Start method of Global.asax. While it's possible to move and use _AppStart as is, we recommend centralizing its logic into Global.asax with existing ASP.NET MVC startup code.
4. Add <roleManager enabled=true /> to the <system.web> section of your root web.config. The Photo Gallery application uses the new WebSecurity membership provider found in WebMatrix.WebData, so we'll need that entry in our configuration for the site to function.
5. Move any stylesheets, script files and images under the Content or Scripts folders in your application. Update any resource links in those files to their new paths.
6. Modify the default Route in Global.asax to point to the Gallery controller and the Default action.
7. Copy the SQL Compact Database found in the App_Data folder to the App_Data Folder of your site. If you're using another database for your site, add that connection string to the Web.Config file in your application.

Figure 2 **Login Controller Actions**

```
public ActionResult Login() {
  ViewBag.Title = "Login";
  return View();
}

[HttpPost]
public ActionResult Login(string email, string password,
  bool? rememberMe, string returnUrl) {
  if (email.IsEmpty())
    ModelState.AddModelError("email",
      "You must specify an email address.");
  if (password.IsEmpty())
    ModelState.AddModelError("password",
      "You must specify a password.");
  if (!ModelState.IsValid)
    return View();
  if (WebSecurity.Login(email, password,
    rememberMe.HasValue ? rememberMe.Value : false)) {
    if (!string.IsNullOrEmpty(returnUrl))
      return Redirect(returnUrl);
    return RedirectToAction("Default", "Gallery");
  }

  ModelState.AddModelError("_FORM",
    "The email or password provided is incorrect");
  return View();
}
```

## Moving from Pages to Views

Once you complete the initial setup of your ASP.NET MVC site, you're ready to migrate the core of your exiting site: the Pages. In Web Pages, a Page (.[cs/vb]html) contains markup, business logic and any data access needed for that page. The main component of your work during a migration to ASP.NET MVC will be to break each page up and divide its content into Controller actions (business logic), data-access classes (data access) and Views (markup).

> The core of the migration process is moving from Pages to Models, Controllers and Views.

First, you need to migrate the Layout of your site. Similar to Master Pages in Web Forms and ASP.NET MVC, Layout pages are files that specify the layout structures of your site. Web Pages and ASP.NET MVC 3 (when used with the Razor view engine) both use the same Layout subsystem, so this portion of the migration should be easy. In the Photo Gallery site, the root _SiteLayout.cshtml file contains our site structure. Copy the contents, then navigate to your ASP.NET MVC site. Open the Layout file located at Views/Shared/_Layout.cshtml and paste in the contents of _SiteLayout.cshtml.

Once completed, you'll need to make a few minor changes to _Layout.cshtml. First, change the link to your stylesheet to the new location in your ASP.NET MVC application (~/Content/Site.css instead of ~/Styles/Site.css). Second, you'll need to change @Page.Title to @ViewBag.Title. Both are objects of type dynamic that can contain display or other data for pages in your site and, as you may have guessed, Page is used for Web Pages, while ViewBag is used for ASP.NET MVC.

The last thing you need to change in your _Layout.cshtml is something you should keep in mind for all of the pages you migrate to ASP.NET MVC. Notice that _Layout.cshtml uses @Href calls to insert URLs into the page. For any call that references static content (scripts, CSS and so on), these can stay unchanged. You will, however, want to change all of the @Href calls that point to pages on your site. While these will also work as is after your migration, they point to static URLs. In ASP.NET MVC, it's considered a better practice to use ASP.NET Routing to create URLs for when Views are rendered. The result is cleaner, less-brittle links are tied to your Route Table definitions, rather than hardcoded on the site.

As such, you'll want to change any links like the following:

```
<div id="banner">
  <p class="site-title">
    <a href="@Href("~/")">Photo Gallery</a>
  </p>
...
</div>
```

Instead, you'll use @Url.RouteUrl or @Url.Action:

```
<div id="banner">
  <p class="site-title">
    <a href="@Url.Action("Default", "Gallery")">Photo Gallery</a>
  </p>
...
</div>
```

Once you've moved your site layout, you can begin to migrate Pages to Views. If, in your Web Pages application, you have any .cshtml pages being executed by RenderPage calls, move those either under Views/Shared for site-wide pages or into the appropriate Views sub-folder for pages shared by a controller, such as Account. Each page that calls one of these partial pages will need to be updated to reflect the new location.

All of your remaining pages should be moved under Views, organized in folders by Controller. Because your Web Pages site has no concept of a Controller, you're going to need to introduce Controllers during a migration. The good news is that a form of Controller structure is evident in the Photo Gallery application, and illustrates a good practice to follow in your own sites.

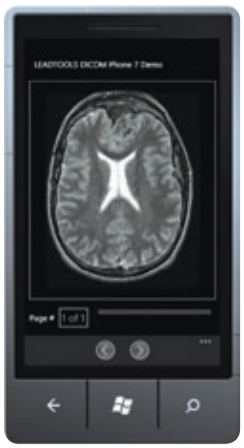
For example, the Photo Gallery template site uses the following folders for grouping pages: Account, Gallery, Photo, Tag and User. Each folder contains pages that enable some functionality related to that grouping. For example, the Account folder contains pages

Figure 3 **Register.cshtml Database Logic**

```
var db = Database.Open("PhotoGallery");

var user = db.QuerySingle("SELECT Email FROM
UserProfiles WHERE LOWER(Email) = LOWER(@0)", email);

if (user == null) {
  db.Execute(
    "INSERT INTO UserProfiles (Email, DisplayName, Bio)
    VALUES (@0, @1, @2)", email, email, "");

  try {
    WebSecurity.CreateAccount(email, password);
    WebSecurity.Login(email, password);
    Response.Redirect("~/");
  } catch (System.Web.Security.MembershipCreateUserException e) {
    ModelState.AddFormError(e.ToString());
  }
} else {
  ModelState.AddFormError("Email address is already in use.");
}
```

# LEADTOOLS® version 17

The World Leader in Imaging SDKs



Powered by Microsoft® Silverlight®

Windows Phone


## Silverlight Imaging SDK

- Load, display and save 100+ formats including PDF, JPEG2000 and TIFF
- Pure managed Silverlight binaries for Silverlight 3, 4 and Windows Phone
- Multi-touch support for Silverlight
- Annotation and markup
- Load, view, process and save DICOM
- Apply visual effects with pixel shaders

The LEADTOOLS Silverlight SDK gives Web and Windows Phone developers the ability to load, display, process, convert and save many different image formats that are not intrinsic to the Silverlight framework. Because of the 100% managed Silverlight binaries, images can be processed and annotated directly within the Silverlight application without the need for external server calls.

The SDK comes with several viewing components including Image Viewer, Image List and Pan Window. The viewer is equipped with automated interactive tools including scroll, pan, scale, zoom and magnifying glass.

Adding Annotations and Markup to your Silverlight application is a cinch with LEADTOOLS fully automated annotations. Load, save, create and edit popular annotation types like line, ruler, polygon, curve, note, highlight, redaction and more.

Create highly interactive medical imaging applications with LEADTOOLS support for DICOM. Load, save and modify tags from DICOM data sets and display 12/16 bpp grayscale images with Window Leveling. LEADTOOLS is currently the only Windows Phone imaging toolkit on the market with support for DICOM data sets.

LEADTOOLS includes sample projects to demonstrate all of the SDK's capabilities including the display of over 100 image formats and incorporation of more than 200 image processing functions.

**Free 60 Day Evaluation!** **www.leadtools.com/msdn**
**800 637-1840**

LEAD TECHNOLOGIES INCORPORATED

Figure 4 **AccountRepository**

```
public class AccountRepository {
  readonly Database _database;
  public AccountRepository() {
    database = Database.Open("PhotoGallery");
  }

  public dynamic GetAccountEmail(string email) {
    return _database.QuerySingle(
      "SELECT Email FROM UserProfiles
      WHERE LOWER(Email) = LOWER(@0)", email);
  }

  public void CreateAccount(string email) {
    _database.Execute(
      "INSERT INTO UserProfiles
      (Email, DisplayName, Bio) VALUES (@0, @1, @2)",
      email, email, "");
  }
}
```

for logging into and out of the site and for registering users. The Gallery folder contains a gallery listing page, a page for adding a new gallery and a page for viewing photos in a gallery. The remaining folders are organized in a similar fashion. While such a structure is not required in Web Pages sites, it does enable an easier migration to ASP.NET MVC. In this case, each folder maps nicely to a Controller and each .cshtml file to an Action and View.

Let's start by moving the Account folder and its three pages—Login, Logout and Register—into your ASP.NET MVC application under the Views directory. In ASP.NET MVC parlance, your Pages instantly become Views by the nature of their location in the application. You're not done, though, because your application needs a Controller and action in order to deliver those Views to the user when requested.

## Introducing Controllers

By MVC convention, the fact that you have an Account folder under Views means you should have a Controller named AccountController, so our next step is to create that Controller under the Controllers folder. Simply right-click and select Add | Controller. From this empty Controller we can create action methods that will contain the logic that now resides at the top of each of the .cshtml pages we moved into our application.

We'll address Login.cshtml first, which contains the code in **Figure 1**.

Notice there are two scenarios being handled here. The first is for when the user loads the login page for the first time. In this scenario, the Page sets its title and transitions directly to markup. The second scenario is contained in the IsPost conditional, and represents the logic that executes when the user completes the login form and clicks the Login button.

In ASP.NET MVC, we handle the process of delivering an empty form and accepting a form submission by creating two action methods in our Controller, one for the empty form and one to handle the submission. The first action will set the page title and return the login View, while the second will contain the logic within the IsPost conditional. These actions are contained in **Figure 2**. After you've added these two actions, delete the header code from Login.cshtml.

There are a number of key differences to note between the original page and the resulting action methods. For starters, you'll notice that the IsPost conditional isn't needed. In ASP.NET MVC, we create a post action for the login page by creating a second Login action method and decorating it with the [HttpPost] attribute. Our first Login method now does nothing more than set the ViewBag.Title property and return a ViewResult, which will then look for a view page in Views/Account called Login.cshtml.

The second thing you might notice is that our Post action contains several parameters and that all of the Request calls that the original page used are gone. By putting parameters on our method that correspond to field names in our Login form (email, password and rememberMe) we can use the ASP.NET MVC default model binder to have those items passed to us as parameters to the action, which saves us from calling the Request object ourselves and makes our action logic more succinct.

Finally, there are some slight differences in how validation is handled and redirects are performed in Web Pages and ASP.NET MVC applications. In our Web Pages site, ModelState.AddError and .AddFormError are the calls we use to notify the page that we've encountered invalid form data. In ASP.NET MVC applications, we use ModelState.AddModelError, which is only slightly different, but a required change for all of your pages. For redirects, our Web Pages site calls Response.Redirect when re-routing the user. In ASP.NET MVC, because our Controller actions should return an ActionResult, we call return RedirectToRoute("Default"), which yields the same result.

> By abstracting our data access into repository classes, we can encapsulate this logic.

Once we've migrated the login page, we can also quickly deal with Logout.cshtml. In Web Pages, some pages may contain logic and no markup if their purpose is to perform an action and then redirect the user, as with Logout.cshtml:

```
@{
  WebSecurity.Logout();
  Response.Redirect("~/");
}
```

In ASP.NET MVC, we'll add a Logout action that performs this work for us:

```
public ActionResult Logout() {
  WebSecurity.Logout();
  return RedirectToAction("Default", "Gallery");
}
```

Because Views represent only the visual elements of a page and no functionality, and we've created an action that handles logging out the user and redirecting them, we can delete the Logout.cshtml View from our application.

So far, we've turned our Account pages into views by copying them into the Views/Account folder, created an AccountController to handle requests to our Account pages, and implemented action methods to handle login and logout scenarios. At this point, you can build and run the site and append Account/Login to the address bar in your

browser (note that the default homepage points to Gallery/Default, which we haven't implemented yet, thus it won't display).

The other piece of site functionality you'll want to deal with at this point is the code and helpers you've contained within the App_Code directory of your Web Pages site. At the beginning of the migration, you can move this entire directory over to your ASP.NET MVC application and include it in your project. If the directory contains any code files (.cs or .vb) you can keep them in App_Code or move them elsewhere. In either case, you'll need to change the Build Action property of each file to Compile rather than Content. If the directory contains .cshtml files with @helper method declarations, you can leave those and utilize them as is in your ASP.NET MVC application.

For the remainder of your Web Pages site, you'll follow a similar cycle of creating a Controller for each Views folder, creating actions for each Page and moving the header code from each page into one or more actions. In no time, you should have all of your pages cleanly separated into Controller actions and Views. However, there's still one piece of the MVC pattern we haven't talked about yet in this article: the Model.

## Migrating Data Access into Repository Classes

Your process for taking the business logic from each page and moving that logic into one or more Controller actions will be pretty straightforward, with one exception: data access. While some of your pages might be similar to the login and logout pages and contain some logic and no data access, much of your Web Pages site probably uses a database.

The Account/Register.cshtml page is one example. When the user completes the registration form and clicks Register, the page makes two database calls, illustrated in **Figure 3**.

Figure 5 **Register Action Using AccountRepository**

```
[HttpPost]
public ActionResult Register(string email, string password,
  string confirmPassword) {

  // Check Parameters (omitted)

  if (!ModelState.IsValid)
    return View();

  var db = new AccountRepository();
  var user = db.GetAccountEmail(email);

  if (user == null) {
    db.CreateAccount(email);

    try {
      WebSecurity.CreateAccount(email, password);
      WebSecurity.Login(email, password);
      return RedirectToAction("Default", "Gallery");
    }
    catch (System.Web.Security.MembershipCreateUserException e) {
      ModelState.AddModelError("_FORM", e.ToString());
    }
  }
  else {
    ModelState.AddModelError("_FORM",
      "Email address is already in use.");
  }

  return View();
}
```

First, the register page opens the PhotoGallery database and returns a WebMatrix.Data.Database object that represents the database. Then the page uses the object to look for an existing e-mail address with the value provided by the user. If the address doesn't exist, a new UserProfile record is created and an account is created for the user using the WebSecurity membership provider.

As long as we've added a reference to WebMatrix.Data and set the Copy Local property to true, we can use this database logic without any changes and the site will function normally. As you're in the midst of migration, this might be the approach you wish to take as a tactical step to keep the site functional.

> You shouldn't arbitrarily change your URLs, even for a migration.

In this article, however, we're going to take things a step further and create additional objects that contain your data access, just as we'd do for an ASP.NET MVC application were we starting from scratch. There are many patterns at your disposal to separate your Controller and data-access logic. We'll use the Repository pattern for the Photo Gallery, and by abstracting our data access into repository classes, we can encapsulate this logic and minimize the impact should we choose to add formal Model objects or an object-relational mapping (ORM) system such as the Entity Framework down the road.

We'll start by creating a Repositories folder in our application, along with a simple class called AccountRepository.cs. Then we can step through each database call in our Register action and move that logic into our repository, as shown in **Figure 4**.

We added the call to Database.Open to the constructor of our repository and created two methods, one for looking up an account e-mail and another for creating the account.

Notice that the return type for GetAccountEmail is dynamic. In WebMatrix.Data, many of the query methods return either dynamic or IEnumerable<dynamic>, and there's no reason you can't continue this practice in your repositories for as long as that practice is sustainable.

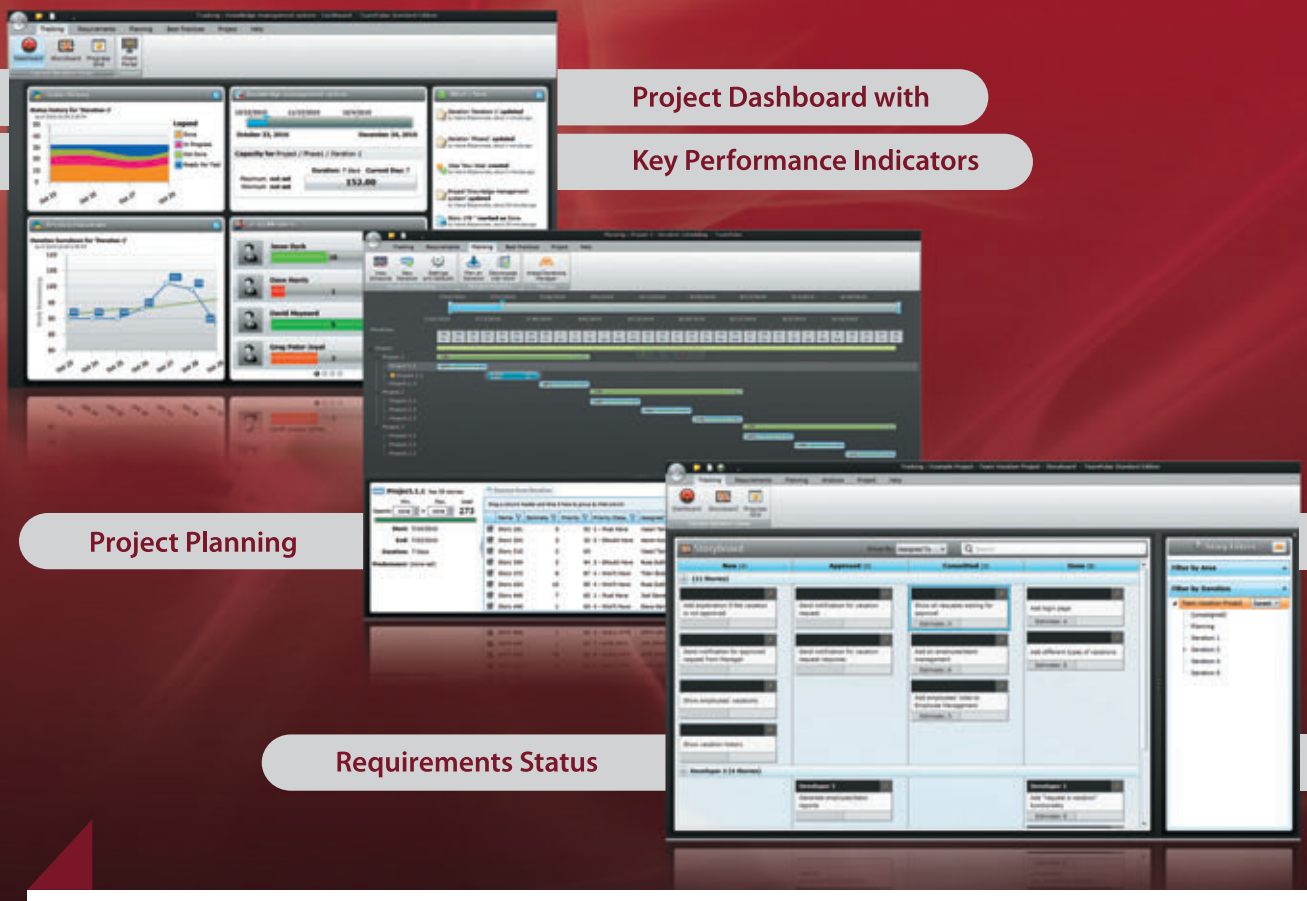
The new Register method—using our AccountRespository—is illustrated in **Figure 5**.

Using dynamic return types is completely acceptable, and might even be wise during a migration as you get your site up and running as a full-fledged ASP.NET MVC application. You aren't required to use strongly typed models in an ASP.NET MVC application, so you can utilize this strategy as long as you don't need a code-driven definition of your data model. The Controller logic and Views of your dynamic-model ASP.NET MVC application will operate as normal, with one exception.

You may have noticed that, in your Web Pages application, form fields are explicitly defined using standard markup:

```
<input type="text" />
<input type="submit" />
...
```

In ASP.NET MVC, the preferred way of using form controls is by using Html helper methods such as Html.TextBox or Html.TextBoxFor because these methods use the Model passed into your View to set current values and handle form validation. If




# *Forget spreadsheets! You deserve better tools to manage your team!*

Meet **Telerik TeamPulse** – a new generation of highly visual project management tools that make it easier for you and your team to manage software development projects. TeamPulse allows your team to be more expressive in the way they capture requirements, manage project plans, assign and track work, and more importantly, be continually connected with each other. Without reams of flat, disconnected and complex spreadsheets TeamPulse supports your entire development lifecycle, from idea to release, by ensuring a constant flow of context, deep insight, and real-time collaboration.

## Try now at: www.telerik.com/pulse

**Exclusive offer for MSDN Magazine readers***
Purchase Telerik TeamPulse online and at checkout, use coupon: TEAMCOU-JSQAKB

**20%off**

**valid until June 30, 2011*

2010 Microsoft Central & Eastern Europe PARTNER OF THE YEAR Winner

telerik
deliver more than expected

you want to use these helper methods in your Views post-migration, you'll have to introduce strongly typed Model objects and move away from using dynamic types in your repositories, because these helper methods can't work with dynamic models.

## Preserving Site URLs

Your site URLs are important. No matter the state of your site, many external sources depend on your existing URLs—search engines, documentation, communications, test scripts and the like. Because of these dependencies, you shouldn't arbitrarily change your URLs, even for a migration.

Consider using ASP.NET Routing to ensure existing URLs are preserved. ASP.NET Routing facilitates a request and matches it to the correct resource, in our case a Controller and Action. Web Pages uses a different routing system than ASP.NET MVC, so you'll need to spend some time ensuring that your existing URLs are preserved.

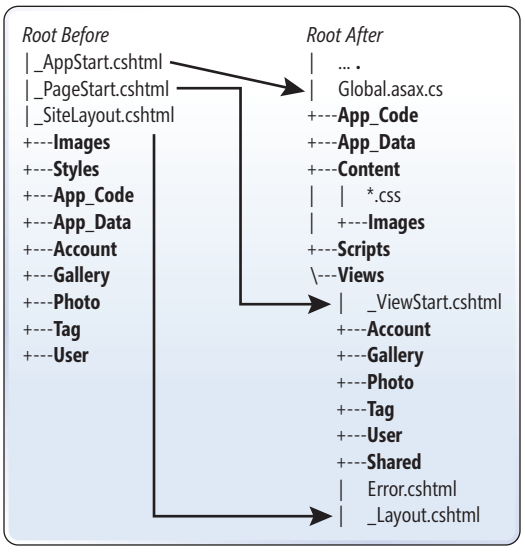



Figure 6 **Application Layout Before and After**

Web Pages applications can handle URLs with and without extensions. For example, both of these URLs resolve to the same page:

```
http://mysite/Gallery/Default.cshtml
http://mysite/Gallery/Default
```

An ASP.NET MVC application, however, won't handle the first URL using the .cshtml extension. Using extension-less URLs throughout your site will ensure that search engines and other dependent sites do the same, which will minimize the migration impact to your site. If, however, you do need to handle existing URLs with extensions, you can create routes in your ASP.NET MVC application to ensure these aren't broken.

For example, consider the default route for our Photo Gallery application:

```
routes.MapRoute(
  "Default",
  "{controller}/{action}/{id}",
  new { controller = "Home",
    action = "Index", id = "" }
);
```

To support legacy URLs in our system, we'll need to add additional routes to our route table above this definition in the Global.asax file. Here's an example of one such definition:

```
routes.MapRoute(
  "LegacyUrl",
  "{controller}/{action}.cshtml/{id}",
  new { controller = "Gallery",
    action = "Default", id = "" }
);
```

In this route entry, URLs that contain the .cshtml extension are handled and sent to the appropriate Controller and Action, assuming your existing Web Pages site structure maps cleanly to a Controller/Action structure.

When planning a migration, keep in mind that your application might require a change to the default route or even additional routes to support existing URLs. If, however, you decide to break an existing URL, be sure to include Actions to handle permanent redirects for users.

## Wrapping Up

To underscore the impact of a migration from Web Pages to ASP.NET MVC, let's take a look at the before and after structure of our Photo Gallery site. In **Figure 6**, you'll see the structure of the Web Pages site in WebMatrix on the left side. The right side shows that site after a completed migration to ASP.NET MVC. Notice that, while there are differences in structure, much of the end result will feel familiar to you.

Today, ASP.NET developers have three framework options to choose from: Web Forms, Web Pages and ASP.NET MVC. While each has its strengths, choosing one doesn't inhibit you from leveraging another or even migrating altogether at some point in the future. And as all three are built on top of ASP.NET, moving from one to the other should never be predicated on technical reasons. If you do choose to move, the similarities between Web Pages and ASP.NET MVC would enable you to continue to use technologies such as NuGet, Razor, Web Deploy, IIS Express and SQL Compact without modification.

> Migrating from Web Pages to ASP.NET MVC is the lowest-friction path.

If you do build your application using Web Pages and decide that a move is a good idea for you, migrating from Web Pages to ASP.NET MVC is the lowest-friction path, especially if you make some up-front design decisions in your Web Pages site to group pages in folders by functionality, use relative URLs for all resources and place all business logic at the top of each page. When the time does come for migration, you'll find that the move from Web Pages to ASP.NET MVC is smooth and straightforward, as intended.

You can find links to many of the techniques and technologies used in this article, plus a lot more, at bit.ly/WebMatrixToMVC. ■

**Brandon Satrom** *works as a senior developer evangelist for Microsoft outside of Austin, Texas. He blogs at userInexperience.com, podcasts at DeveloperSmackdown.com and can be followed on Twitter at twitter.com/BrandonSatrom.*

**Clark Sell** *works as a senior developer evangelist for Microsoft outside of Chicago. He blogs at csell.net, podcasts at DeveloperSmackdown.com and can be followed on Twitter at twitter.com/csell5.*

**Thanks** to the following technical experts for reviewing this article: *Phil Haack and Erik Porter*

# Build Interactive Reports with ZERO Code. Deliver Them to All .NET Platforms.


Telerik Reporting is the first .NET solution to offer developers the ability to easily deliver interactive and consistent reports to the most used Microsoft platforms - Windows Azure, SharePoint 2010, Silverlight, WPF, ASP.NET and Windows Forms. The tool offers a completely codeless experience and swift performance, fueled by a robust OLAP engine.

- Interactive reports in any .NET business application
- Powerful OLAP Data Processing Engine
- Extensive database support (including cubes)
- Intuitive WYSIWYG report designer with countless wizards, builders, previews and tools
- Numerous export formats and end-user features

**Try now at: www.telerik.com/InteractiveReports**

2010 Microsoft Central & Eastern Europe PARTNER OF THE YEAR Winner

telerik
*deliver more than expected*

# DynWaitList: ID-Based Windows Event Multiplexing

Alex Gimenez

Microsoft Windows provides multiplexed listening to several events via the WaitForMultipleObjects method and its variants. These functions are powerful, but inconvenient to use when the event list is dynamic.

The difficulty is that event signals are identified by *indexes* into an array of object handles. Such indexes will shift when events are added to or removed from the middle of the array.

This type of problem is commonly addressed by having a container that stores the handles, wrapping the array and performing insertion, removal and lookups on behalf of client applications.

This article discusses the design and implementation of such a container class. The container stores the event handles used with the WaitForMultipleObjects method. Users of the container class refer to individual handles by a numeric ID that won't change for the lifetime of the container, even as events are added or removed.

This article discusses:

- Exploring the issue
- Using DynWaitList
- Common uses of DynWaitList
- Design and performance considerations
- Adding a handle
- Removing a handle
- Detecting signals
- Multi-core considerations

Technologies discussed:

Windows API

Code download available at:

code.msdn.microsoft.com/mag201105DynWaitList

## Exploring the Issue

The interface to WaitForMultipleObjects/MsgWaitForMultipleObjects is best suited for the simpler cases where:

- You know beforehand how many handles you're going to wait on.
- The number of handles you're waiting on doesn't change over time.

> When a handle is signaled, you get the index of the handle as a return value.

When a handle is signaled, you get the index of the handle as a return value. This index—the position within the event array passed as input—isn't directly meaningful. What you really want out of these functions is the handle that was signaled or some non-ephemeral information that leads you to that handle.

**Figure 1** shows an example that illustrates the issue. Its code—part of a theoretical media-streaming application—is waiting for a signal from an audio device or from the network (this and other code samples can be found in the code download for this article).

Getting a result of WAIT_OBJECT_0 (index 0) means the audio device was signaled. Getting an index of 1 means the network was

Figure 1 **Media-Streaming App Waiting for a Signal**

```
#define MY_AUDIO_EVENT (WAIT_OBJECT_0)
#define MY_SOCKET_EVENT (WAIT_OBJECT_0 + 1)
HANDLE handles[2];
handles[0] = audioInHandle;
handles[1] = socketHandle;
...
switch( WaitForMultipleObjects(handles) )
{
  case MY_AUDIO_EVENT:
  // Handle audio event
  break;
  case MY_SOCKET_EVENT:
  // Handle socket event
  // What happens if we need to stop audio here?
  break;
}
```

signaled. Now, what happens if you need to close audioInHandle in response to an event triggered from socketHandle? You would then have to get rid of index 0 in the handles array, which would shift the indexes greater than 0—meaning that the MY_SOCKET_EVENT value needs to be dynamic, not a constant.

There are ways to work around this situation, of course (for example, keep the optional handles at the end of the array, or shift the start of the array), but they get messy quickly when you add a few more events and the handling of error paths (indexes off of WAIT_ABANDONED_0).

At first glance, the issue is that you can't use constants to identify the event handles. Looking closer, we see that the root problem with this interface is that it uses array indexes to identify the event handles. Indexes then play an inconvenient double-duty here, representing both the handle's position in memory and the fact that the event is signaled.

Figure 2 **Using DynWaitList**

```
WORD idAudioEvent, idSocketEvent;
DynWaitList<10> handles(100);  // Max 10 events, first ID is 100

handles.Add( socketHandle,  &idSocketEvent );
handles.Add( audioInHandle, &idAudioEvent  );
...
switch( handles.Wait(2000)  )
{
  case (HANDLE_SIGNALED| idAudioEvent ):
  // Handle audio event
  break;
  case (HANDLE_SIGNALED| idSocketEvent):
  // Handle socket event
  if( decided_to_drop_audio )
  {
    // Array will shift within; the same ID
    // can be reused later with a different
    // handle if, say, we reopen audio
    handles.Remove(idAudioEvent);

    // Any value outside the
    // 100...109 range is fine
    idAudioEvent = 0;
  }
  break;

  case (HANDLE_ABANDONED| idSocketEvent):
  case (HANDLE_ABANDONED| idAudioEvent):
  // Critical error paths
  break;

  case WAIT_TIMEOUT:
  break;
}
```

It would be nice if the signaled events were identifiable independently from the indexes in the array. That's what the DynWaitList class does for us.

## Using DynWaitList

The DynWaitList class is a container list for the array of handles to be passed for the WaitForMultipleObjects method. The internal collection of handles has a static maximum size. The class is implemented as a template, where the maximum size of the collection is the only template parameter.

The container interface has the methods you'd expect: Add to insert an event and specify its ID, Remove to remove an event and a few variants of Wait. **Figure 2** shows how DynWaitList is used to solve the issue presented earlier.

## Common Uses of DynWaitList

The example presented here shows a small number of well-known event IDs. There are, however, cases where the IDs are many, and aren't known beforehand. These are some common cases:

- A TCP server, which would hold an event ID for each currently connected client socket. This is the case that makes the most of the dynamic event list, as client sockets do come and go with each connection.
- An audio mixing application or IP phone application, which would have a handle to wait for the frame-ready/timer signal of each audio device on the system.

It would be nice if the signaled events were identifiable independently from the indexes in the array.

The examples so far show a common theme: The dynamic list of handles is representative of the changing external environment around the application.

## Design and Performance Considerations

Implementing a container is a balancing act of picking between conflicting goals of performance, simplicity and storage space. These have to be evaluated under the light of the most frequent container uses—the ones shown earlier. Then, it helps to enumerate the operations to perform on the container and their rate of occurrence:

- **Adding a handle:** quite frequent
- **Removing a handle:** about the same frequency as adding a handle
- **Changing a handle:** not applicable (you can't change the handle of an existing object in Windows)
- **Translating the container into the flat array that Windows needs:** quite frequent
- **Retrieving the value of a handle that was just signaled:** quite frequent

Figure 3 **Class Declaration Showing Main Interface and Container Internals**

```
class DynWaitlistImpl
{
  protected:
    DynWaitlistImpl( WORD nMaxHandles, HANDLE *pHandles,
      WORD *pIds, WORD wFirstId );

    // Adds a handle to the list; returns TRUE if successful
    BOOL Add( HANDLE hNewHandle, WORD *pwNewId );

    // Removes a handle from the list; returns TRUE if successful
    BOOL Remove( WORD wId );

    DWORD Wait(DWORD dwTimeoutMs, BOOL bWaitAll = FALSE);

    // ... Some snipped code shown later ...

  private:
    HANDLE *m_pHandles;
    WORD *m_pIds;
    WORD m_nHandles;
    WORD m_nMaxHandles;
};

template <int _nMaxHandles> class DynWaitlist: public DynWaitlistImpl
{
  public:
    DynWaitlist(WORD wFirstId):
    DynWaitlistImpl( _nMaxHandles, handles, ids, wFirstId ) { }
    virtual ~DynWaitlist() { }

  private:
    HANDLE handles[ _nMaxHandles ];
    WORD ids[ _nMaxHandles ];
};
```

With these operations in mind, I decided to have internal storage be an array of event handles (the one required by Windows), plus a parallel array of IDs, which are 16-bit values. This parallel-array arrangement allows for efficient translation between indexes and event IDs. Specifically:

- The array that Windows needs is always available.
- Given the index returned by Windows, looking up its ID is an order-1 operation.

One other important consideration is thread safety. Given the purpose of this container, it's fair to require the operations to be serialized, so I chose not to protect the internal arrays.

**Figure 3** shows the declaration of the class showing the main interface and container internals.

Notice how the class is broken in two, with a base class holding an array pointer, and a template-derived class holding the actual storage. This provides flexibility for dynamic array allocation—if needed—by deriving a different template class. This implementation uses solely static storage.

## Adding a Handle

Adding a handle to the array is straightforward, except for the act of finding a free ID to represent a newly created event. Per the chosen design, there's an array of IDs in the container. This array is pre-allocated to hold up to the maximum number of IDs the container can hold. Thus, the array can conveniently hold two groups of IDs:

- The first *N* elements are the IDs *in use*, where *N* is how many handles are actually allocated.
- The remaining elements are a pool of free IDs.

This requires the ID array to be filled with all possible ID values upon construction. Given that, it's trivial to find a free ID using the ID *just past* the last one used. No search is required. The code for the class constructor and the Add method is shown in **Figure 4**. These two methods cooperate to build and use the pool of free IDs.

> Implementing a container is a balancing act of picking between conflicting goals of performance, simplicity and storage space.

## Removing a Handle

To remove a handle from the container given its ID, we must find the handle's index. The translation from index to ID is optimized to order-1 by this implementation, but this comes at the expense of the reverse translation's performance. Given an ID, it takes a linear search (order-N) to find its index in the array. I decided to take that hit because, in the case of a server, users aren't as mindful of response time upon disconnecting. After finding the index to remove, the removal operation is quick and simple—just swap the handle found with the last "in-use" handle (see **Figure 5**).

Figure 4 **Class Constructor and Add Method**

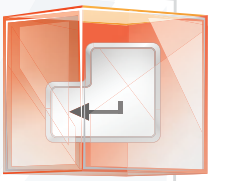
```
DynWaitlistImpl::DynWaitlistImpl(
  WORD nMaxHandles,  // Number of handles
  HANDLE *pHandles,   // Pointer to array of handle slots
  WORD *pIds,         // Pointer to array of IDs
  WORD wFirstID)      // Value of first ID to use
// Class Constructor. Initializes object state
:  m_nMaxHandles(nMaxHandles)
,  m_pHandles(pHandles)
,  m_pIds(pIds)
,  m_nHandles(0)
{
  // Fill the pool of available IDs
  WORD wId = wFirstID;
  for( WORD i = 0; i < nMaxHandles; ++i )
  {
    m_pIds[i] = wId;
    wId++;
  }
}


BOOL DynWaitlistImpl::Add(
  HANDLE hNewHandle, // Handle to be added
  WORD *pwNewId ) // OUT parameter - value of new ID picked
// Adds one element to the array of handles
{
  if( m_nHandles >= m_nMaxHandles )
  {
    // No more room, no can do
    return FALSE;
  }
  m_pHandles[ m_nHandles ] = hNewHandle;

  // Pick the first available ID
  (*pwNewId) = m_pIds[ m_nHandles ];

  ++m_nHandles;
  return TRUE;
}
```

devart

www.devart.com

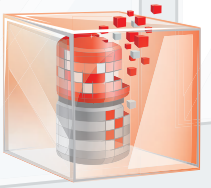
**Query Builder**
Create any queries visually, without code typing

**SQL Complete**
Code autocomplete tool with SQL formatting for SQL Server Management Studio



**Schema Compare**
Compare and synchronize SQL Server schemas

**Data Studio**
Explore and analyze large data sets in SQL Server databases



**Data Compare**
Compare and synchronize table data of SQL Server

# dbForge for SQL Server

## Must-have SQL Server development and management tools

## Detecting Signals

Detecting signals is the main work performed by DynWaitList. Calling WaitForMultipleObjects is trivial because all the data is kept pre-archived for the call. Translating the signal detected into an ID that the upper layer can reference is also trivial because of the parallel array of IDs. That code is the meat of the *Wait* method, shown in **Figure 6**. There are a few variations of *Wait*, all of which use the internal TranslateWaitResult method to perform the index-to-ID translation.

> I believe event multiplexing is important even in computers with multiple cores.

## Multi-Core Considerations

We're marching toward a "manycore" computing world, where we extract efficiency from computers by performing work in multiple threads. In that world, is event multiplexing even important? Could it be that most applications will end up having one event per thread, thus neutralizing the benefits of DynWaitList?

I bet not. I believe event multiplexing is important even in computers with multiple cores, at least because of these two reasons:

- Some operations simply don't benefit from parallelism, because they touch hardware devices that must be accessed serially. Low-level networking is one such example.
- One key benefit of event multiplexing—especially in utility libraries—is not to push a particular threading model upon an application. The top-level application should dictate the threading model. In this manner, the application should be free to *choose* the distribution of events into its threads, making the encapsulation of event wait lists even more important.

## Simple Code, Fewer Bugs

Summing up, associating non-ephemeral IDs to each event passed to Windows WaitForMultipleObjects functions can lead to code simplification and reduce the likelihood of bugs, as it removes the burden placed upon the application to translate meaningless event indexes into useful object handles or pointers. The DynWaitList class efficiently encapsulates this association process into a wrapper around these Windows API wait functions. All operations involved are of order-1, except for construction and handle removal, which are of order-N. Further optimization can be achieved by sorting the array, which would add a slight penalty to handle addition while making handle removal much faster. ■

Figure 5 **The Removal Operation**

```
BOOL DynWaitlistImpl::Remove(
  WORD wId ) // ID of handle being removed
// Removes one element from the array of handles
{
  WORD i;
  BOOL bFoundIt = FALSE;
  for( i = 0; i < m_nHandles; ++i )
  {
    // If we found the one we want to remove
    if( m_pIds[i] == wId )
    {
      // Found it!
      bFoundIt = TRUE;
      break;
    }
  }

  // Found the ID we were looking for?
  if( bFoundIt )
  {
    WORD wMaxIdx = (m_nHandles - 1);
    if( i < wMaxIdx ) // if it isn't the last item being removed
    {
      // Take what used to be the last item and move it down,
      // so it takes the place of the item that was deleted
      m_pIds    [i] = m_pIds    [ wMaxIdx ];
      m_pHandles[i] = m_pHandles[ wMaxIdx ];

      // Save the ID being removed, so it can be reused in a future Add
      m_pIds    [ wMaxIdx ] = wId;
    }
    --m_nHandles;
    m_pHandles[m_nHandles] = 0;
    return TRUE;
  }
  else
  {
    return FALSE;
  }
}
```

Figure 6 **Detecting Signals**

```
// Snippet from the header file - Wait is a quick, inline method
DWORD Wait(DWORD dwTimeoutMs, BOOL bWaitAll = FALSE)
{
  return TranslateWaitResult(
    WaitForMultipleObjects( m_nHandles,
    m_pHandles,
    bWaitAll,
    dwTimeoutMs )
    );
}

// Snippet from the CPP file - method called by all flavors of Wait
DWORD DynWaitlistImpl::TranslateWaitResult(
  DWORD dwWaitResult ) // Value returned by WaitForMultipleObjectsXXX
  // translates the index-based value returned by Windows into
  // an ID-based value for comparison
{

  if( (dwWaitResult >= WAIT_OBJECT_0) &&
    (dwWaitResult < (DWORD)(WAIT_OBJECT_0 + m_nHandles) ) )
  {
    return HANDLE_SIGNALED | m_pIds[dwWaitResult - WAIT_OBJECT_0];
  }
  if( (dwWaitResult >= WAIT_ABANDONED_0) &&
    (dwWaitResult < (DWORD)(WAIT_ABANDONED_0 + m_nHandles) ) )
  {
    return HANDLE_ABANDONED | m_pIds[dwWaitResult - WAIT_ABANDONED_0];
  }

  return dwWaitResult; // No translation needed for other values
}
```

**ALEX GIMENEZ** *currently works as a developer near Seattle, writing real-time audio/video/networking software with the team responsible for delivering Microsoft Lync. His prior experience spans just about two decades dealing with point-of-sale, embedded, device-driver and telecommunications software. In his spare time, he enjoys biking, drumming and studying Japanese. He can be reached at alexgim@microsoft.com.*

**THANKS** to the following technical experts for reviewing this article: *Bart Holmberg, Warren Lam and Jiannan Zheng*

# Microsoft Platform Ready

## Speed your time to market


# Get your application ready for the cloud with no-cost technical support.

Planning your future with cloud services? You can access training, support, testing, and marketing resources to help you take your solution to market faster.

**Join Microsoft Platform Ready and receive:**

- Free Developer Resources and Tools
- Access to Free Technical Support via Phone, Email and Online Chat.
- Special offers to take advantage of Windows Azure.
- Free Online Testing Tools to help get your application compatible with Windows Azure.
- Marketing support and use of the "Powered by Windows Azure" logo.

Visit **www.microsoftplatformready.com** and sign up today >>

## Get your FREE Windows Azure platform 30 Day Pass

Try out the Windows Azure platform at no cost. No credit card required. Use this FREE pass to learn about Windows Azure and SQL Azure. Get plenty of capacity to experiment with.

Visit **http://bit.ly/WindowsAzurePass** and use Promo Code **MSDNJ1** to get started today!

Follow Us @MSPlatformReady

Find Us @Microsoftplatformready

# Visualizing Bing Routes on Windows Phone 7

Sandrino Di Mattia

Microsoft Windows Phone 7 comes with an easy-to-use geolocation API that allows you to determine the current position and movement of a user, expressed in latitude and longitude (and sometimes also the altitude). Once you have access to this data, you're ready to create location-aware features in your Windows Phone 7 application.

If you're building an application for a hamburger restaurant, it would be great if—besides presenting the menu and promotions—you could also locate the nearest restaurant based on a user's current location. Another great feature would be the ability to find people near you. Think about scenarios for salespeople who want to see if they can visit their clients in-between meetings.

This article will focus on how to bring this data to a Windows Phone 7 app and how to visualize routes and locations in different ways. The actual data comes from the Bing Maps API. So before I can show you some advanced concepts, it's important to take a look at the basics of the Bing Maps API.

This article discusses:

- Introduction to the Bing Maps API
- Calculating the route
- Showing the route on a map
- Showing directions
- Finding your current position
- Real applications and performance
- Marketplace submission
- A sample application

Technologies discussed:

Bing Maps API, Windows Phone 7

Code download available at:

code.msdn.microsoft.com/mag201105Bing

## Introduction to the Bing Maps API

The first thing you need is a working Bing account. On the Bing Maps Account Center (bingmapsportal.com), you need to use a Windows Live ID to sign up for a Bing Maps account. After creating your account, you'll have access to your account details, where it will be possible to create a key. Because you'll be building an application for Windows Phone 7, you're allowed to write something like http://localhost for the Application URL.

Besides creating an account, this page also allows you to monitor the usage for the Bing Maps API. Should you decide to use Bing Maps in a production application, you'll also need to come back to this page to contact someone regarding licensing.

The Bing Maps API actually provides a few services, and the Windows Phone 7 application will be consuming the SOAP services. I'll run through short overviews of the following services:

- **Geocode** dev.virtualearth.net/webservices/v1/geocodeservice/geocodeservice.svc
- **Imagery** dev.virtualearth.net/webservices/v1/imageryservice/imageryservice.svc
- **Route** dev.virtualearth.net/webservices/v1/routeservice/routeservice.svc
- **Search** dev.virtualearth.net/webservices/v1/searchservice/searchservice.svc

The Geocode service allows you to work with coordinates and addresses; the Imagery service will let you work with actual images (aerial, bird's-eye and road); the Route service will help you calculate the route between two or more points; and the Search service allows you to look for locations based on human input (such as "restaurant in Brussels").

To make use of these services, you only need to add a "Service Reference" to one of the previous URLs. Note that this action will create or update the ServiceReferences.ClientConfig file. **Figure 1** shows an example of how you can invoke the Geocode method of the Geocode service.

Each service method is based on request/response. You create a request object where you prepare the question for the server and configure the API key. In this case, I created a GeocodeRequest and I'll be asking the server to provide me with the GeocodeLocation of "Brussels, Belgium." After creating the request, I just invoke the client in an asynchronous way. And finally, you'll always receive a response that will provide you with information—and in case of problems, will also find the error. Run the sample application in the accompanying download to view the GeocodeServiceClient in action and see how the location (or error) is displayed on-screen using data binding.

The application will be using the Geocode and the Route services to calculate the route between two addresses and show them to the user.

## Calculating the Route

Using the Bing Route service, you can calculate the route from point A to point B. Just like the previous example, this works with a request/response. You'll need to find the actual geolocation of each address first (using a GeocodeRequest), and using these locations you'll be able to create a RouteRequest. The sample application in the accompanying download contains all the code, but **Figure 2** shows a short example of how this is done.

> There are many scenarios where an application would benefit from knowing the user's location.

Note that the Waypoints property of the request is a collection that allows you to add multiple waypoints. This can be interesting when you need to know the route for a complete itinerary instead of just from point A to point B.

When you execute the CalculateRouteAsync method, the service will start doing the hard work: calculating the route; listing all itinerary items (actions such as turning, taking an exit and so on); calculating duration and distance; listing all points (geolocations) and more. **Figure 3** shows an overview of some important data present in the RouteResponse.

Figure 1 **Invoking the Geocode Method of the Geocode Service**

```
// Create the request.
var geoRequest = new GeocodeRequest();
geoRequest.Credentials = new Credentials();
geoRequest.Credentials.ApplicationId = "<my API key>";
geoRequest.Address = new Address();
geoRequest.Address.CountryRegion = "Belgium";
geoRequest.Address.PostalTown = "Brussels";

// Execute the request and display the results.
var geoClient = new GeocodeServiceClient("BasicHttpBinding_IGeocodeService");
geoClient.GeocodeAsync(geoRequest);
geoClient.GeocodeCompleted += (s, e) =>
{
  if (e.Result != null && e.Result.Results.Any(o =>
    o.Locations != null && o.Locations.Any()))
    Location = e.Result.Results.FirstOrDefault().Locations.FirstOrDefault();
  else if (e.Error != null)
    Error = e.Error.Message;
  else
    Error = "No results or locations found.";
};
```

> Using the Bing Route service, you can calculate the route from point A to point B.

## Showing the Route on a Map

In my first example, I'll be using the RoutePath Points to show the route on a map. Because the Windows Phone 7 Toolkit already includes the Bing Maps control, you'll only need to add a reference to the assembly Microsoft.Phone.Controls.Maps. After that it's easy to display the map on the phone. Here's an example of how to display the map showing Brussels (the CredentialsProvider is required to set the API key):

```
<maps:Map Center="50.851041,4.361572" ZoomLevel="10"
  CredentialsProvider="{StaticResource MapCredentials}" />
```

If you intend to use any of the controls in the Maps assembly, I advise you to add a reference to this assembly before adding a

Figure 2 **Creating a RouteRequest**

```
// Create the request.
var routeRequest = new RouteRequest();
routeRequest.Credentials = new Credentials();
routeRequest.Credentials.ApplicationId = "<my API key>";
routeRequest.Waypoints = new ObservableCollection<Waypoint>();
routeRequest.Waypoints.Add(fromWaypoint);
routeRequest.Waypoints.Add(toWaypoint);
routeRequest.Options = new RouteOptions();
routeRequest.Options.RoutePathType = RoutePathType.Points;
routeRequest.UserProfile = new UserProfile();
routeRequest.UserProfile.DistanceUnit = DistanceUnit.Kilometer;

// Execute the request.
var routeClient = new RouteServiceClient("BasicHttpBinding_IRouteService");
routeClient.CalculateRouteCompleted +=
  new EventHandler<CalculateRouteCompletedEventArgs>(OnRouteComplete);
routeClient.CalculateRouteAsync(routeRequest);
```

service reference to the Bing services. The service reference will then reuse types such as Microsoft.Phone.Controls.Maps.Platform.Location instead of creating new types, and you won't need to write conversion methods or value converters to use some of the data returned by the service.

So now you know how to calculate the route between two points and how to display a map on the phone. Let's bring these two techniques together to visualize the route on the map. The Points in the RoutePath will be used to draw on the map. The Map control allows you to add shapes such as a Pushpin (to show the start and end of the route, for example) and a MapPolyline (to draw the route based on GeoCoordinates).

Because the points returned by the service aren't of the same type as the points used by the Maps control, I've created two small extension methods to convert the points to the correct type, shown in **Figure 4**.

You can use these extension methods on the RouteResponse when the CalculateRoute method completes. After conversion, these extension methods will return types that can, for example, be used for binding to the Bing Maps control. Because this is a Silverlight application, we should use an IValueConverter to do the actual conversion. **Figure 5** shows an example of the value converter that will convert the Locations to GeoCoordinates.

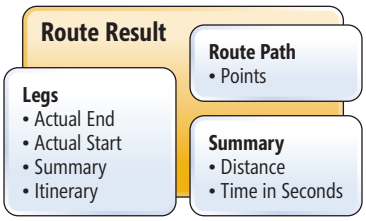



Figure 3 **RouteResponse Content**

Now it's time to configure the data binding. The data binding will use the converters, so it's important to declare those first. These can be declared in the page resources or in the application resources (if you plan to reuse the converters), as shown here:

```
<phone:PhoneApplicationPage.Resources>
  <converters:LocationConverter x:Key="locationConverter" />
  <converters:ItineraryItemDisplayConverter x:Key="itineraryConverter" />
</phone:PhoneApplicationPage.Resources>
```

After declaring the converters you can add the maps control and other overlay controls (like the MapPolyline and Pushpin) and bind them to the required properties, as shown here:

```
<maps:Map Center="50.851041,4.361572" ZoomLevel="10"
  CredentialsProvider="{StaticResource MapCredentials}">
  <maps:MapPolyline Locations="{Binding RoutePoints,
    Converter={StaticResource locationConverter}}"
    Stroke="#FF0000FF" StrokeThickness="5" />
  <maps:Pushpin Location="{Binding StartPoint,
    Converter={StaticResource locationConverter}}" Content="Start" />
  <maps:Pushpin Location="{Binding EndPoint,
    Converter={StaticResource locationConverter}}" Content="End" />
</maps:Map>
```

Because the Windows Phone 7 Toolkit already includes the Bing Maps control, you'll only need to add a reference to the assembly Microsoft.Phone.Controls.Maps.

As you can see, these bindings use the converters that were declared previously to convert the data into a format that's understood by the maps control. Finally, you need to set these properties after the CalculateMethod completes, as shown here:

```
private void OnRouteComplete(object sender,
CalculateRouteCompletedEventArgs e)
{
  if (e.Result != null && e.Result.Result != null
    && e.Result.Result.Legs != null & e.Result.Result.Legs.Any())
  {
    var result = e.Result.Result;
    var legs = result.Legs.FirstOrDefault();

    StartPoint = legs.ActualStart;
    EndPoint = legs.ActualEnd;
    RoutePoints = result.RoutePath.Points;
    Itinerary = legs.Itinerary;
  }
}
```

**Figure 6** shows the screen after launching the application and calculating the route.

## Showing Directions

As you can see, displaying the route on a map is pretty standard stuff. In the next example, I'll show how you can build a custom

Figure 4 **Extension Methods to Convert Points to Correct Type**

```
public static GeoCoordinate ToCoordinate(this Location routeLocation)
{
  return new GeoCoordinate(routeLocation.Latitude, routeLocation.Longitude);
}

public static LocationCollection ToCoordinates(this IEnumerable<Location> points)
{
  var locations = new LocationCollection();

  if (points != null)
  {
    foreach (var point in points)
    {
      locations.Add(point.ToCoordinate());
    }
  }

  return locations;
}
```

Figure 5 **Using an IValueConverter**

```
public class LocationConverter : IValueConverter
{
  public object Convert(object value, Type targetType,
    object parameter, CultureInfo culture)
  {
    if (value is Location)
    {
      return (value as Location).ToCoordinate();
    }
    else if (value is IEnumerable<Location>)
    {
      return (value as IEnumerable<Location>).ToCoordinates();
    }
    else
    {
      return null;
    }
  }
}
```

# SEARCH LESS. DEVELOP MORE.

## WITH THE INTELLIGENT LIBRARY IN THE CLOUD


Stop by our
**TechEd booth #1815,**
next to **O'Reilly Media**

# REGISTER >YOUR TEAM FOR A FREE TRIAL NOW

Find all the latest and most relevant resources for Microsoft Developers and IT professionals at Safari Books Online.

**LEARN MORE AT >** **safaribooksonline.com/msdn-teched**
...and get access to the world's most popular, fully searchable digital library.

Safari® Books Online

Move your library to the cloud and get instant, unlimited, fully searchable access to the tech books you need – including exclusive online access to books from O'Reilly Media, Addison-Wesley, Prentice Hall and more!

See for yourself why more than 15 million IT and business professional, developers, and web designers from corporations, government agencies and academic institutions use Safari Books Online!

control that displays the directions from start to end using the text and summary of each ItineraryItem. **Figure 7** shows the final result.

In **Figure 1**, you see that Legs is one of the RouteResult properties. The Legs property contains one or more "Leg" objects, each of which includes a collection of ItineraryItems. Using the ItineraryItems, it will be possible to fill up the control that you can see in **Figure 7**. Each line in **Figure 7** shows an ItineraryItem with the total seconds and total distance for that step, and the index of the current step. The ItineraryItem doesn't keep track of the current step count, so I created a small class called ItineraryItemDisplay:

```
public class ItineraryItemDisplay
{
  public int Index { get; set; }
  public long TotalSeconds { get; set; }
  public string Text { get; set; }
  public double Distance { get; set; }
}
```

The sample code in the accompanying download also contains an extension method with the following signature:

```
public static ObservableCollection
  <ItineraryItemDisplay> ToDisplay(this
  ObservableCollection<ItineraryItem> items)
```

The code in this method loops through all items, writes the important values to a new ItineraryItemDisplay object and also keeps track of the current step count in the Index property. Finally, the ItineraryItemDisplayConverter takes care of the conversion during the data binding. As you might have noticed in **Figure 7**, each itinerary step is nicely formatted (cities and streets are marked in bold) using a custom control called ItineraryItemBlock. Its only goal is to format the ItineraryItem text in a clean way. In **Figure 7**, you can also see a blue block with some extra information, but this is regular data binding:

```
[TemplatePart(Name = "ItemTextBlock", Type =
  typeof(TextBlock))] public class
  ItineraryItemBlock : Control
```

The TemplatePart attribute defines an element that should be present in the control Template, and also what type that element should be. In this case, it should be a TextBlock called ItemTextBlock:

```
<Style TargetType="controls:ItineraryItemBlock"
x:Key="ItineraryItemBlock">
  <Setter Property="Template">
    <Setter.Value>
      <ControlTemplate TargetType=
        "controls:ItineraryItemBlock">
        <TextBlock x:Name="ItemTextBlock"
          TextWrapping="Wrap"
          LineStackingStrategy=
          "BlockLineHeight" LineHeight="43" />
      </ControlTemplate>
    </Setter.Value>
  </Setter>
</Style>
```

The reason for choosing a TextBlock is obvious. Using the TextBlock Inlines property, you can add content to the TextBlock in code. The OnApplyMethod can be overriden in a custom control, and this is when you'll want to get ahold of the ItemTextBlock (see **Figure 8**).

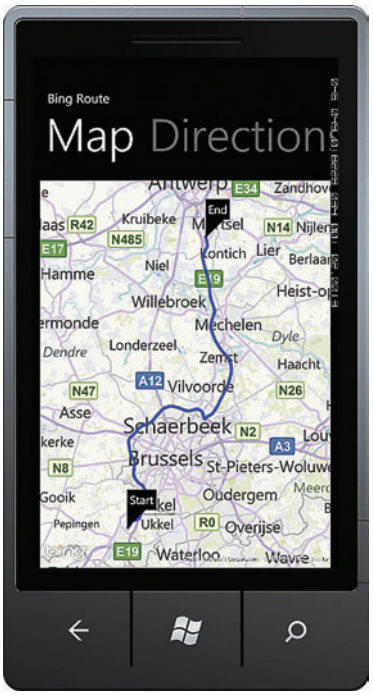



Figure 6 **Visual Representation of the Route**

The ItineraryItem Text property will be dissected and used to fill up this TextBlock with some extra formatting. Actually, this is easy to do because the Text property contains little more than regular text. Some parts are surrounded with XML tags:

```
<VirtualEarth:Action>Turn</VirtualEarth:Action>
<VirtualEarth:TurnDir>left</VirtualEarth:TurnDir>onto
<VirtualEarth:RoadName>Guido Gezellestraat
</VirtualEarth:RoadName>
```

Because I only want to highlight cities and road names, I wrote a small method that strips tags such as VirtualEarth:Action or VirtualEarth:TurnDir. After retrieving the TextBlock, the method SetItinerary is called, and this is where the Inlines are added to the TextBlock (see **Figure 9**).

As you can see in the preceeding XML text example, not every part of the text is contained in an XML element. To be able to use this text in an XmlReader, the first thing to do is to wrap it in a dummy XML element. This allows us to create a new XmlReader with this text. Using the XmlReader Read method, you can loop each part of the XML string.

Based on the NodeType, you can figure the current position in the XML string. For example, consider the following element: <VirtualEarth:RoadName>Guido Gezellestraat</VirtualEarth:RoadName>. Using the XmlReader Read method, you'll have three iterations. The first one is <VirtualEarth:RoadName> and this is an XmlNodeType.Element. Because the goal is to format the roads and cities in bold, this is where a Run object is added to the TextBlock Inlines with the FontWeight set to bold. Adding a Run object to the Inlines just appends some text to the TextBlock.

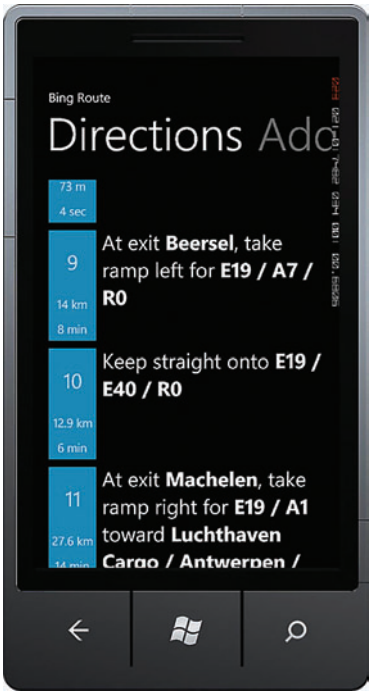



Figure 7 **Displaying Start-to-End Directions**

> The ItineraryItem doesn't keep track of the current step count.

In any other case, you don't need formatting, and this is where you'll want to add a normal Run object containing only text without any formatting properties set.

That's it for the custom control. The whole ItineraryItemDisplay record is displayed using a custom DataTemplate for the ListBox. This DataTemplate also contains a reference to the custom control (see **Figure 10**).

# Powerful Tools for Developers

v4.5!


## High-Performance PDF Printer Driver

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

v4.5!


## PDF Editor for .NET, now Webform Enabled

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

New!


## PDF Integration into Silverlight Applications

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

## OCR Module available with royalty-free licensing!


**The new OCR Module from Amyuni enables developers to:**

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

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

**Learn more at www.amyuni.com**

Windows 7

Certified for Windows Server 2008

## More Development Tools Available at:

# www.amyuni.com



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

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

AMYUNI Technologies

Figure 8 **Finding the TextBlock**

```
/// <summary>
/// When the template is applied, find the textblock.
/// </summary>
public override void OnApplyTemplate()
{
  base.OnApplyTemplate();

  // Get textblock.
  textBlock = GetTemplateChild("ItemTextBlock") as TextBlock;
  if (textBlock == null)
    throw new InvalidOperationException
      ("Unable to find 'ItemTextBlock' TextBlock in the template.");

  // Set the text if it was assigned before loading the template.
  if (!String.IsNullOrEmpty(Text))
    SetItinerary(Text);
}
```

Now that the custom control and styling are ready, the only task left to do is to implement this in the Pivot control and in the code. As I mentioned before, the custom control and DataTemplate will be used in a ListBox:

```
<controls:PivotItem Header="Directions">
  <ListBox ItemsSource=
    "{Binding Itinerary, Converter={StaticResource itineraryConverter}}"
    Grid.RowSpan="2" ItemTemplate="{StaticResource ItineraryItemComplete}" />
</controls:PivotItem>
```

This ListBox ItemSource is bound to the Itinerary property, and this is how the property is populated; afterward, the ItineraryItemDisplayConverter does the rest. As you can see, using a custom control and a bit of styling, you can take the itinerary data from the Route service and make it appealing to the user:

Figure 9 **Adding Inlines to TextBlock with the SetItinerary Method**

```
// Read the input
string dummyXml = String.Format(
  "<Itinerary xmlns:VirtualEarth=\"http://dummy\">{0}</Itinerary>",
  itinerary);
using (var stringReader = new StringReader(dummyXml))
{
  // Trace the previous element.
  string previousElement = "";

  // Parse the dummy xml.
  using (var xmlReader = XmlReader.Create(stringReader))
  {
    // Read each element.
    while (xmlReader.Read())
    {
      // Add to textblock.
      if (!String.IsNullOrEmpty(xmlReader.Value))
      {
        if (previousElement.StartsWith("VirtualEarth:"))
        {
          textBlock.Inlines.Add(new Run()
            { Text = xmlReader.Value, FontWeight = FontWeights.Bold });
        }
        else
        {
          textBlock.Inlines.Add(new Run() { Text = xmlReader.Value });
        }
      }

      // Store the previous element.
      if (xmlReader.NodeType == XmlNodeType.Element)
        previousElement = xmlReader.Name;
      else
      previousElement = "";
    }
  }
}
```

```
private void OnRouteComplete(object sender, CalculateRouteCompletedEventArgs e)
{
  if (e.Result != null && e.Result.Result != null &&
    e.Result.Result.Legs != null & e.Result.Result.Legs.Any())
  {
    ...
    Itinerary = e.Result.Result.Legs.FirstOrDefault().Itinerary;
  }
}
```

## Finding Your Current Position

In the previous examples, you learned how to use the Geocode and Route services to get directions from point A to point B and how to visualize these directions. Now it's time to look at the geolocation API.

GeoCoordinateWatcher is the class you'll use to find out the current GPS coordinates:

```
coordinateWatcher = new GeoCoordinateWatcher
  (GeoPositionAccuracy.High); coordinateWatcher.StatusChanged +=
  new EventHandler<GeoPositionStatusChangedEventArgs>(OnCoordinateUpdate);
coordinateWatcher.Start();
```

If you're building an application that displays directions, you could use the position changes to scroll through each step automatically and even play a sound.

The GeoCoordinateWatcher will go through different stages after executing the Start method, but when the status is set to Ready, you'll have access to the current position. Best practice is to call the Stop method after you're done working with the GeoCoordinateWatcher:

```
private void OnCoordinateStatusChanged(object sender,
  GeoPositionStatusChangedEventArgs e)
{
  if (e.Status == GeoPositionStatus.Ready)
  {
    coordinateWatcher.Stop();

    // Get position.
    fromLocation = coordinateWatcher.Position.Location;
    LocationLoaded();
  }
}
```

Now the sample application also provides location-aware features. The GeoCoordinateWatcher also exposes a PositionChanged event that allows you to monitor when the position changes. If you're building an application that displays directions, you could use the position changes to scroll through each step automatically and even play a sound based on the VirtualEarth:Action in the ItineraryItem Text. You'll end up having an actual GPS navigation application.

Are you debugging the application using the Windows Phone 7 emulator? If you're testing your application's geolocalization functionalities, you might bump into a small problem with the GeoCoordinateWatcher Status: It will always stay on NoData and

never change to Ready. That's why it's important to write your code against the interface (IGeoPositionWatcher<GeoCoordinate>) and not the implementation (GeoCoordinateWatcher). In Tim Heuer's blog post (bit.ly/cW4fM1), you can download the EventListGeoLocationMock class that simulates an actual GPS device.

The EventListGeoLocationMock class accepts a collection of GeoCoordinateEventMocks that should simulate the user's coordinates in time. This will allow you to test the user's location and movement:

```
GeoCoordinateEventMock[] events = new GeoCoordinateEventMock[]
{
  new  GeoCoordinateEventMock { Latitude = 50, Longitude = 6,
    Time = new TimeSpan(0,0,5) },
  new  GeoCoordinateEventMock { Latitude = 50, Longitude = 7,
    Time = new TimeSpan(0,15,0) }
};

IGeoPositionWatcher<GeoCoordinate>  coordinateWatcher =
  new EventListGeoLocationMock(events); coordinateWatcher.StatusChanged +=
  new EventHandler<GeoPositionStatusChangedEventArgs>(...);
coordinateWatcher.Start();
```

Based on the device name, you could determine if the application is running on a real device or on the emulator to decide which IGeoPositionWatcher to use. Look for the extended property "DeviceName," which is always set to XDeviceEmulator when running the application in the emulator:

```
private static bool IsEmulator()
{
  return (Microsoft.Phone.Info.DeviceExtendedProperties.GetValue("DeviceName")
    as string) == "XDeviceEmulator";
}
```

Alternatively, on Dragos Manolescu's blog (bit.ly/h72vXj), you can find another way to mock Windows Phone 7 event streams using Reactive Extensions, or Rx.

Figure 10 **The Entire ItineraryItemDisplay Record in a Custom DataTemplate for the Listbox**

```
<!-- Template for a full item (includes duration and time) -->
<DataTemplate x:Key="ItineraryItemComplete">
  <Grid Height="173" Margin="12,0,12,12">
    <!-- Left part: Index, Distance, Duration. -->
    <Grid HorizontalAlignment="Left" Width="75">
      <Grid.ColumnDefinitions>
        <ColumnDefinition Width="25*" />
        <ColumnDefinition Width="25*" />
        <ColumnDefinition Width="25*" />
        <ColumnDefinition Width="25*" />
      </Grid.ColumnDefinitions>
      <Grid.RowDefinitions>
        <RowDefinition Height="50*"></RowDefinition>
        <RowDefinition Height="20*"></RowDefinition>
        <RowDefinition Height="20*"></RowDefinition>
      </Grid.RowDefinitions>

      <!-- Gray rectangle. -->
      <Rectangle Grid.ColumnSpan="4" Grid.RowSpan="3" Fill="#FF0189B4" />

      <!-- Metadata fields. -->
      <TextBlock Text="{Binding Index}"
        Style="{StaticResource ItineraryItemMetadata}"
        Grid.Row="0" Grid.Column="1" Grid.ColumnSpan="2" />
      <TextBlock Text="{Binding Distance,
        Converter={StaticResource kilometers}}"
        Style="{StaticResource ItineraryItemMetadata}"
        FontSize="{StaticResource PhoneFontSizeSmall}"
        Grid.Row="1" Grid.Column="0" Grid.ColumnSpan="4" />
      <TextBlock Text="{Binding TotalSeconds,
        Converter={StaticResource seconds}}"
        Style="{StaticResource ItineraryItemMetadata}"
        FontSize="{StaticResource PhoneFontSizeSmall}"
        Grid.Row="2" Grid.Column="0" Grid.ColumnSpan="4" />
    </Grid>

    <!-- Right part to show directions. -->
    <StackPanel Margin="84,-4,0,0" VerticalAlignment="Top" >
      <controls:ItineraryItemBlock Text="{Binding Text}"
        Style="{StaticResource ItineraryItemBlock}"
        FontSize="{StaticResource PhoneFontSizeLarge}"
        Foreground="{StaticResource PhoneForegroundBrush}"
        Padding="0,3,0,0" Margin="0,0,0,5" />
    </StackPanel>
  </Grid>
</DataTemplate>
```

> Don't forget that your application is running on a mobile device and a regular Wi-Fi connection isn't always available.

## Real Applications and Performance

When you're building an application and you want it to sell, obviously it should be appealing to the user. The user will want a fast application that has great features. The previous examples showed that you'll need to call a few Web service methods and handle a few asynchronous events before you can show some results to the user. Don't forget that your application is running on a mobile device and a regular Wi-Fi connection isn't always available.

Reducing the Web service calls and data going over the wire could speed up your application. In the introduction, I talked about a restaurant application that provides the menu and location-aware features. If you're building such an application, there could be a service running in a cloud that provides menus and promotions to the phone. Why not use this service to perform the complex calculations instead of running them on the phone? Here's an example:

```
[ServiceContract]
public interface IRestaurantLocator
{
  [OperationContract]
  NearResult GetNear(Location location);
}
```

You could build a service that uses the user's current location. The service will start some threads (the example uses Parallel.ForEach) and calculate the distance between this location and other restaurants simultaneously (see **Figure 11**).

Looping through each restaurant in the restaurants list in parallel, the location of each restaurant will be converted to a geolocation using the GeocodeServiceClient. Using this location and the user's location, the route is calculated between these points using the RouteServiceClient. Finally, the TotalSeconds property of the route summary is used to find the three nearest restaurants, and those are sent to the device.

The advantage here is that calculations are run at the same time (using Parallel.ForEach and depending on the machine's resources), and once they're done, only the relevant data goes to the Windows Phone. Performance-wise, you'll feel the difference; the mobile

# we are Countersoft
# you are Control


## GEMINI
### Project Platform

**The most versatile project management platform for software development and testing teams around the world**

The award-winning Gemini Project Platform is the .NET based project platform that uniquely allows teams to work the way they want to work with more functionality and easy customization for optimum team performance and predictable results.

**BOOK A DEMO / FREE TRIAL / 3-for-FREE**
Seeing is believing! We'll talk about YOU.
www.**geminiplatform**.com

## ATLAS
### Product Support Optimized

**The first comprehensive solution to address ISVs' profitability AND customer experience by targeting product support**

Patent pending Atlas addresses the area of the software business that matters most: the product support eco-system. Atlas is the logical alternative to revenue-sapping, customer-irritating forums. Atlas offers smarter community-based support and integrates FAQ/KB, documentation and how-to videos.

**DOWNLOAD NOW**
and lead from the front.
www.**atlasanswer**.com

COUNTERSOFT ENABLING COLLECTIVE CAPABILITY www.countersoft.com

Contact // Europe/Asia: +44 (0)1753 824000 // US/Canada: 800.927.5568 // E: sales@countersoft.com

Figure 11 **Calculating Distance Between a User's Location and Three Nearby Restaurants**

```
public NearResult GetNear(BingRoute.Location location)
{
  var near = new NearResult();
  near.Restaurants = new List<RestaurantResult>();

  ...

  Parallel.ForEach(restaurants, (resto) =>
  {
    try
    {
      // Build geo request.
      var geoRequest = new BingGeo.GeocodeRequest();
      ...

      // Get the restaurant's location.
      var geoResponse = geoClient.Geocode(geoRequest);

      // Restaurant position.
      if (geoResponse.Results.Any())
      {
        var restoLocation =
          geoResponse.Results.FirstOrDefault().Locations.FirstOrDefault();
        if (restoLocation != null)
        {
          // Build route request.
          var fromWaypoint = new Waypoint();
          fromWaypoint.Description = "Current Position";
          ...;

          var toWaypoint = new Waypoint();
          ...

          // Create the request.
          var routeRequest = new RouteRequest();
          routeRequest.Waypoints = new Waypoint[2];
          routeRequest.Waypoints[0] = fromWaypoint;
          routeRequest.Waypoints[1] = toWaypoint;
          ...

          // Execute the request.
          var routeClient = new RouteServiceClient();
          var routeResponse = routeClient.CalculateRoute(routeRequest);

          // Add the result to the result list.
          if (routeResponse.Result != null)
          {
            var result = new RestaurantResult();
            result.Name = resto.Name;
            result.Distance = routeResponse.Result.Summary.Distance;
            result.TotalSeconds = routeResponse.Result.Summary.TimeInSeconds;
            results.Add(result);
          }
        }
      }
    }
    catch (Exception ex)
    {
      // Take appropriate measures to log the error and/or show it to the end user.
    }
  });


  // Get the top 3 restaurants.
  int i = 1;
  var topRestaurants = results.OrderBy(o => o.TotalSeconds)
                              .Take(3)
                              .Select(o => { o.Index = i++; return o; });

  // Done.
  near.Restaurants.AddRange(topRestaurants);
  return near;

}
```

application will only call a single Web service method and only a little data goes over the wire.

Besides that, the code and asynchronous calls on the Windows Phone 7 are reduced dramatically, as demonstrated here:

```
var client = new RestaurantLocatorClient();
client.GetNearCompleted += new EventHandler<
  GetNearCompletedEventArgs>(OnGetNearComplete);
client.GetNearAsync(location);
```

**Figure 12** shows the display of nearby restaurants on the phone.

> Reducing the Web service calls and data going over the wire could speed up your application.

## Marketplace Submission

The last thing I want to mention is the Windows Phone 7 Marketplace submission process. Your application needs to pass a set of requirements to be allowed on the Marketplace. One of these requirements is defining the capabilities of your application in the application manifest file. If you decide to use the GeoCoordinateWatcher, you'll also need to define the ID_CAP_LOCATION capability in the application manifest file.

The MSDN Library page, "How to: Use the Capability Detection Tool for Windows Phone" (bit.ly/hp7fjG), explains how to use the Capability Detection Tool to detect all capabilities used by your application. Be sure to read through the article before you submit your application!

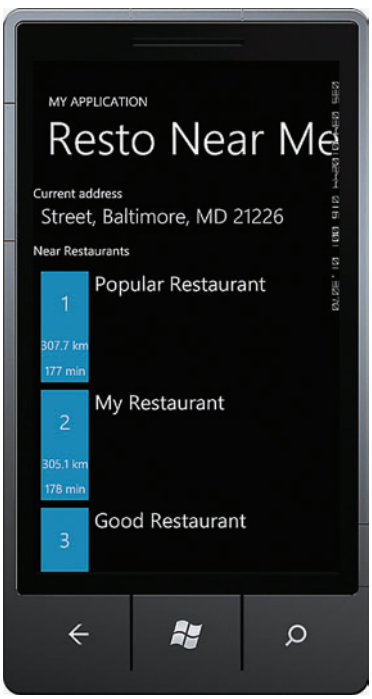



Figure 12 **The Phone Display of Three Nearby Restaurants**

## Sample Application

The code download for this application contains a solution with two projects. One is a class library that contains the control, extension methods and styles that you can easily integrate into your own projects. The second is a sample Windows Phone 7 Pivot application that combines all the examples in a small application. ■

**Sandrino Di Mattia** *is a Microsoft enthusiast. In his job as technical consultant at RealDolmen, he integrates Microsoft technologies and products to create solutions that work for customers and their businesses. In his spare time, he participates in Belgian user groups and writes articles on his blog, blog.sandrinodimattia.net.*

**Thanks** to the following technical expert for reviewing this article: *Dragos Manolescu*

# Code First in the ADO.NET Entity Framework 4.1

Rowan Miller

The ADO.NET Entity Framework 4.1 was released back in April and includes a series of new features that build on top of the existing Entity Framework 4 functionality that was released in the Microsoft .NET Framework 4 and Visual Studio 2010.

The Entity Framework 4.1 is available as a standalone installer (msdn.microsoft.com/data/ee712906), as the "EntityFramework" NuGet package and is also included when you install ASP.NET MVC 3.01.

The Entity Framework 4.1 includes two new main features: DbContext API and Code First. In this article, I'm going to cover how these two features can be used to develop applications. We'll take a quick look at getting started with Code First and then delve into some of the more advanced capabilities.

The DbContext API is a simplified abstraction over the existing ObjectContext type and a number of other types that were included in previous releases of the Entity Framework. The DbContext API surface is optimized for common tasks and coding patterns. Common functionality is exposed at the root level and more advanced functionality is available as you drill down through the API.

Code First is a new development pattern for the Entity Framework that provides an alternative to the existing Database First and Model First patterns. Code First lets you define your model using CLR classes; you can then map these classes to an existing database or use them to generate a database schema. Additional configuration can be supplied using Data Annotations or via a fluent API.

This article discusses:

- Getting started
- Change tracker API
- Deployment considerations
- More to learn

Technologies discussed:

Entity Framework 4.1

## Getting Started

Code First has been around for a while, so I'm not going to go into detail on getting started. You can complete the Code First Walkthrough (bit.ly/evXl0c) if you aren't familiar with the basics. **Figure 1** is a complete code listing to help get you up and running with a Code First application.

For the sake of simplicity, I'm choosing to let Code First generate a database. The database will be created the first time I use BlogContext to persist and query data. The rest of this article will apply equally to cases where Code First is mapped to an existing database schema. You'll notice I'm using a database initializer to drop and recreate the database as we change the model throughout this article.

## Mapping with the Fluent API

Code First begins by examining your CLR classes to infer the shape of your model. A series of conventions are used to detect things such as primary keys. You can override or add to what was

Figure 1 **Getting Started with Code First**

```
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;
using System;

namespace Blogging
{
  class Program
  {
    static void Main(string[] args)
    {
      Database.SetInitializer<BlogContext>(new BlogInitializer());

      // TODO: Make this program do something!
    }
  }

  public class BlogContext : DbContext
  {
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
      // TODO: Perform any fluent API configuration here!
    }
  }

  public class Blog
  {
    public int BlogId { get; set; }
    public string Name { get; set; }
    public string Abstract { get; set; }

    public virtual ICollection<Post> Posts { get; set; }
  }

  public class RssEnabledBlog : Blog
  {
    public string RssFeed { get; set; }
  }

  public class Post
  {
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }
    public byte[] Photo { get; set; }

    public virtual Blog Blog { get; set; }
  }

  public class BlogInitializer : DropCreateDatabaseIfModelChanges<BlogContext>
  {
    protected override void Seed(BlogContext context)
    {
      context.Blogs.Add(new RssEnabledBlog
      {
        Name = "blogs.msdn.com/data",
        RssFeed = "http://blogs.msdn.com/b/data/rss.aspx",
        Posts = new List<Post>
        {
          new Post { Title = "Introducing EF4.1" },
          new Post { Title = "Code First with EF4.1" },
        }
      });

      context.Blogs.Add(new Blog { Name = "romiller.com" });
      context.SaveChanges();
    }
  }
}
```

detected by convention using Data Annotations or a fluent API. There are a number of articles about achieving common tasks using the fluent API, so I'm going to look at some of the more advanced configuration that can be performed. In particular, I'm going to focus on the "mapping" sections of the API. A mapping configuration can be used to map to an existing database schema or to affect the shape of a generated schema. The fluent API is exposed via the DbModelBuilder type and is most easily accessed by overriding the OnModelCreating method on DbContext.

The Entity Framework 4.1 includes two new main features: DbContext API and Code First.

**Entity Splitting** Entity splitting allows the properties of an entity type to be spread across multiple tables. For example, say I want to split the photo data for posts out into a separate table so that it can be stored in a different file group. Entity splitting uses multiple Map calls to map a subset of properties to a specific table. In **Figure 2**, I'm mapping the Photo property to the "PostPhotos" table and the remaining properties to the "Posts" table. You'll notice that I didn't include the primary key in the list of properties. The primary key is always required in each table; I could have included it, but Code First will add it in for me automatically.

**Table-per-Hierarchy (TPH) Inheritance** TPH involves storing the data for an inheritance hierarchy in a single table and using a discriminator column to identify the type of each row. Code First will use TPH by default if no configuration is supplied. The discriminator column will be aptly named "Discriminator" and the CLR type name of each type will be used for the discriminator values.

You may, however, want to customize how TPH mapping is performed. To do this, you use the Map method to configure the discriminator column values for the base type and then Map<TEntityType> to configure each derived type. Here I'm using a "HasRssFeed" column to store a true/false value to distinguish between "Blog" and "RssEnabledBlog" instances:

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
{
  modelBuilder.Entity<Blog>()
    .Map(m => m.Requires("HasRssFeed").HasValue(false))
    .Map<RssEnabledBlog>(m => m.Requires("HasRssFeed").HasValue(true));
}
```

In the preceeding example, I'm still using a standalone column to distinguish between types, but I know that RssEnabledBlogs can

Figure 2 **Entity Splitting**

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
{
  modelBuilder.Entity<Post>()
    .Map(m =>
      {
        m.Properties(p => new { p.Title, p.Content });
        m.ToTable("Posts");
      })
    .Map(m =>
      {
        m.Properties(p => new { p.Photo });
        m.ToTable("PostPhotos");
      });
}
```

Figure 3 **Combining Entity Splitting with TPT Inheritance Mapping**

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
{
  modelBuilder.Entity<Blog>()
    .Map(m =>
      {
        m.Properties(b => new { b.Name });
        m.ToTable("Blogs");
      })
    .Map(m =>
      {
        m.Properties(b => new { b.Abstract });
        m.ToTable("BlogAbstracts");
      })
    .Map<RssEnabledBlog>(m =>
      {
         m.ToTable("RssBlogs");
      });
}
```

be identified by the fact that they have an RSS feed. I can rewrite the mapping to let the Entity Framework know that it should use the column that stores "Blog.RssFeed" to distinguish between types. If the column has a non-null value, it must be an RssEnabledBlog:

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
{
  modelBuilder.Entity<Blog>()
    .Map<RssEnabledBlog>(m => m.Requires(b => b.RssFeed).HasValue());
}
```

**Table-per-Type (TPT) Inheritance** TPT involves storing all properties from the base type in a single table. Any additional properties for derived types are then stored in separate tables with a foreign key back to the base table. TPT mapping uses a Map call to specify the base table name and then Map<TEntityType> to configure the table for each derived type. In the following example, I'm storing data that's common to all blogs in the "Blogs" table and data specific to RSS-enabled blogs in the "RssBlogs" table:

```
modelBuilder.Entity<Blog>()
  .Map(m => m.ToTable("Blogs"))
  .Map<RssEnabledBlog>(m => m.ToTable("RssBlogs"));
```

**Table-per-Concrete Type (TPC) Inheritance** TPC involves storing the data for each type in a completely separate table with no foreign key constraints between them. The configuration is similar to TPT mapping, except you include a "MapInheritedProperties" call when configuring each derived type. MapInheritedProperties lets Code First know to remap all properties that were inherited from the base class to new columns in the table for the derived class:

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
{
  modelBuilder.Entity<Blog>()
    .Map(m => m.ToTable("Blogs"))
    .Map<RssEnabledBlog>(m =>
      {
        m.MapInheritedProperties();
        m.ToTable("RssBlogs");
      });
}
```

By convention, Code First will use identity columns for integer primary keys. However, with TPC there's no longer a single table containing all blogs that can be used to generate primary keys. Because of this, Code First will switch off identity when you use TPC mapping. If you're mapping to an existing database that has been set up to generate unique values across multiple tables, you can re-enable identity via the property configuration section of the fluent API.

**Hybrid Mappings** Of course, the shape of your schema isn't always going to conform to one of the patterns that I've covered, especially if you're mapping to an existing database. The good news is that the mapping API is composable and you can combine multiple mapping strategies. **Figure 3** includes an example that shows combining Entity Splitting with TPT Inheritance Mapping. The data for Blogs is split between "Blogs" and "BlogAbstracts" tables, and the data specific to RSS-enabled blogs is stored in a separate "RssBlogs" table.

## Change Tracker API

Now that I've looked at configuring database mappings, I want to spend some time working with data. I'm going to delve straight into some more advanced scenarios; if you aren't familiar with basic data access, take a minute to read through the Code First Walkthrough mentioned earlier.

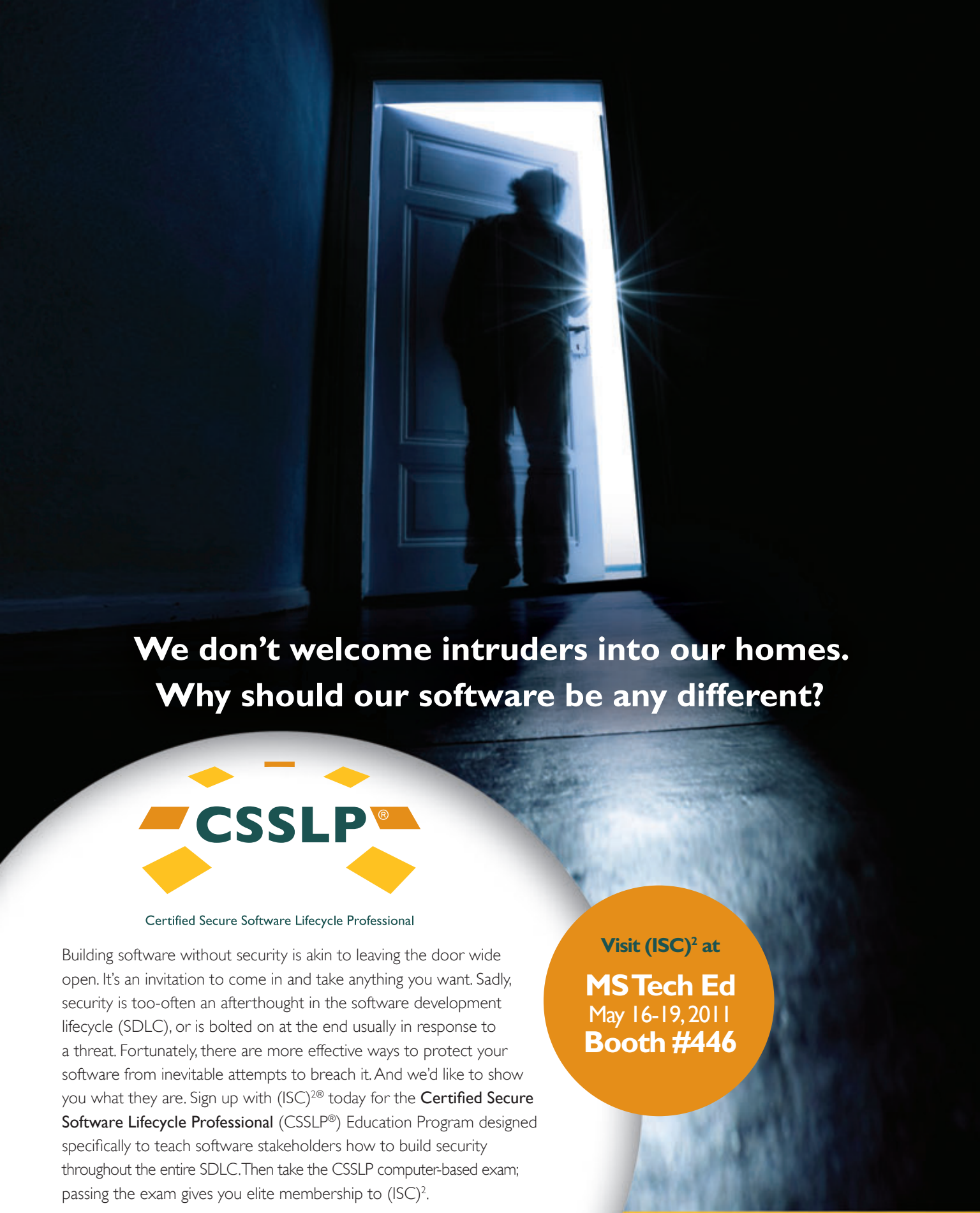
**State Information for a Single Entity** In many cases, such as logging, it's useful to get access to the state information for an entity. This can include things such as the state of the entity and which properties are modified. DbContext provides access to

Figure 4 **Getting State Information for an Entity**

```
static void Main(string[] args)
{
  Database.SetInitializer<BlogContext>(new BlogInitializer());

  using (var db = new BlogContext())
  {
    // Change the name of one blog
    var blog = db.Blogs.First();
    blog.Name = "ADO.NET Team Blog";

    // Print out original and current value for each property
    var propertyNames = db.Entry(blog).CurrentValues.PropertyNames;
    foreach (var property in propertyNames)
    {
      System.Console.WriteLine(
        "{0}\n Original Value: {1}\n Current Value: {2}",
        property,
        db.Entry(blog).OriginalValues[property],
        db.Entry(blog).CurrentValues[property]);
    }
  }

  Console.ReadKey();
}
```

Figure 5 **Accessing Information for Multiple Entities with DbContext**

```
static void Main(string[] args)
{
  Database.SetInitializer<BlogContext>(new BlogInitializer());

  using (var db = new BlogContext())
  {
    // Load all blogs into memory
    db.Blogs.Load();

    // Change the name of one blog
    var blog = db.Blogs.First();
    blog.Name = "ADO.NET Team Blog";

    // Print out state for each blog that is in memory
    foreach (var entry in db.ChangeTracker.Entries<Blog>())
    {
      Console.WriteLine("BlogId: {0}\n State: {1}\n",
        entry.Entity.BlogId,
        entry.State);
    }
  }

  Console.ReadKey();
}
```



# We don't welcome intruders into our homes. Why should our software be any different?

## CSSLP®

Certified Secure Software Lifecycle Professional

Building software without security is akin to leaving the door wide open. It's an invitation to come in and take anything you want. Sadly, security is too-often an afterthought in the software development lifecycle (SDLC), or is bolted on at the end usually in response to a threat. Fortunately, there are more effective ways to protect your software from inevitable attempts to breach it. And we'd like to show you what they are. Sign up with (ISC)²® today for the **Certified Secure Software Lifecycle Professional** (CSSLP®) Education Program designed specifically to teach software stakeholders how to build security throughout the entire SDLC. Then take the CSSLP computer-based exam; passing the exam gives you elite membership to (ISC)².

*Want to slam the door on would-be intruders?*
*Go to www.isc2.org/csslp right now.*

**Visit (ISC)² at**

**MS Tech Ed**
May 16-19, 2011
**Booth #446**

*Connect with us:*
*www.isc2intersec.com*
*www.twitter.com/isc2*
*www.facebook.com/csslp*

(ISC)²®

Figure 6 **Running LINQ Queries Against In-Memory Data**

```
static void Main(string[] args)
{
  Database.SetInitializer<BlogContext>(new BlogInitializer());

  using (var db = new BlogContext())
  {
    // Load all blogs into memory
    db.Blogs.Load();

    // Query for blogs ordered by name
    var orderedBlogs = from b in db.Blogs.Local
                       orderby b.Name
                       select b;

    Console.WriteLine("All Blogs:");
    foreach (var blog in orderedBlogs)
    {
      Console.WriteLine(" - {0}", blog.Name);
    }

    // Query for all RSS enabled blogs
    var rssBlogs = from b in db.Blogs.Local
                   where b is RssEnabledBlog
                   select b;

    Console.WriteLine("\n Rss Blog Count: {0}", rssBlogs.Count());
  }

  Console.ReadKey();
}
```

this information for an individual entity via the "Entry" method. The code snippet in **Figure 4** loads one "Blog" from the database, modifies a property and then prints out the current and original values for each property to the console.

When the code in **Figure 4** is run, the console output is as follows:

```
BlogId
 Original Value: 1
 Current Value: 1

Name
 Original Value: blogs.msdn.com/data
 Current Value: ADO.NET Team Blog

Abstract
 Original Value:
 Current Value:

RssFeed
 Original Value: http://blogs.msdn.com/b/data/rss.aspx
 Current Value: http://blogs.msdn.com/b/data/rss.aspx
```

**State Information for Multiple Entities** DbContext allows you to access information about multiple entities via the "ChangeTracker.Entries" method. There's both a generic overload that gives entities of a specific type and a non-generic overload that gives all entities. The generic parameter doesn't need to be an entity type. For example, you could get entries for all loaded objects that implement a specific interface. The code in **Figure 5** demonstrates loading all blogs into memory, modifying a property on one of them and then printing out the state of each tracked blog.

When the code in **Figure 5** is run, the console output is as follows:

```
BlogId: 1
 State: Modified

BlogId: 2
 State: Unchanged
```

**Querying Local Instances** Whenever you run a LINQ query against a DbSet, the query is sent to the database to be processed. This guarantees that you always get complete and up-to-date results, but if you know that all the data you need is already in memory, you can avoid a round-trip to the database by querying the local data. The code in **Figure 6** loads all blogs into memory and then runs two LINQ queries for blogs that don't hit the database.

When the code in **Figure 6** is run, the console output is as follows:

```
All Blogs:
 - blogs.msdn.com/data
 - romiller.com

 Rss Blog Count: 1
```

**Navigation Property as a Query** DbContext allows you to get a query that represents the contents of a navigation property for a given entity instance. This allows you to shape or filter the items you want to bring into memory and can avoid bringing back unnecessary data.

> Entity splitting allows the properties of an entity type to be spread across multiple tables.

For example, I have an instance of blog and want to know how many posts it has. I could write the code shown in **Figure 7**, but it's relying on lazy loading to bring all the related posts back into memory just so that I can find the count.

That's a lot of data being transferred from the database and taking up memory compared to the single integer result I really need.

Fortunately, I can optimize my code by using the Entry method on DbContext to get a query representing the collection of posts associated with the blog. Because LINQ is composable, I can chain on the "Count" operator and the entire query gets pushed to the database so that only the single integer result is returned (see **Figure 8**).

## Deployment Considerations

So far I've looked at how to get up and running with data access. Now let's look a little further ahead at some things to consider as your app matures and you approach a production release.

**Connection Strings:** So far I've just been letting Code First generate a database on localhost\SQLEXPRESS. When it comes time to deploy my application, I probably want to change the database that Code First is pointed at. The recommended approach for this is to add a connection string entry to the App.config file (or Web.config for Web applications). This is also the recommended approach for using Code First to map to an existing

Figure 7 **Getting a Count of Database Items with Lazy Loading**

```
static void Main(string[] args)
{
  Database.SetInitializer<BlogContext>(new BlogInitializer());

  using (var db = new BlogContext())
  {
    // Load a single blog
    var blog = db.Blogs.First();

    // Print out the number of posts
    Console.WriteLine("Blog {0} has {1} posts.",
      blog.BlogId,
      blog.Posts.Count());
  }

  Console.ReadKey();
}
```

# Our Name Says It All

activePDF®

Come and see why thousands of customers have trusted us over the last decade for all their server based PDF development needs.

. Convert over 400 files types to PDF
. High fidelity translation from PDF to Office
. ISO 32000 Support including PDF/X & PDF/A
. HTML5 to PDF
. True PDF Print Server
. Form Fill PDF
. Append, Stamp, Secure and Digitally Sign

activePDF®
Leading the iPaper Revolution

Compatible with Windows 7 | Windows Vista® | Microsoft Partner Silver Midmarket Solution Provider

**Download our FREE evaluation from www.activepdf.com/MSDN**

Call 1-866-GoTo-PDF | 949-582-9002 | Sales@activepdf.com

Figure 8 **Using DbContext to Optimize Query Code and Save Resources**

```
static void Main(string[] args)
{
  Database.SetInitializer<BlogContext>(new BlogInitializer());

  using (var db = new BlogContext())
  {
    // Load a single blog
    var blog = db.Blogs.First();

    // Query for count
    var postCount = db.Entry(blog)
      .Collection(b => b.Posts)
      .Query()
      .Count();

    // Print out the number of posts
    Console.WriteLine("Blog {0} has {1} posts.",
      blog.BlogId,
      postCount);
  }

  Console.ReadKey();
}
```

database. If the connection string name matches the fully qualified type name of the context, Code First will automatically pick it up at run time. However, the recommended approach is to use the DbContext constructor that accepts a connection name using the name=<connection string name> syntax. This ensures that Code First will always use the config file. An exception will be thrown if the connection string entry can't be found. The following example shows the connection string section that could be used to affect the database that our sample application targets:

```
<connectionStrings>
  <add
    name="Blogging"
    providerName="System.Data.SqlClient"
    connectionString="Server=MyServer;Database=Blogging;
    Integrated Security=True;MultipleActiveResultSets=True;" />
</connectionStrings>
```

Here's the updated context code:

```
public class BlogContext : DbContext
{
  public BlogContext()
    : base("name=Blogging")
  {}

  public DbSet<Blog> Blogs { get; set; }
  public DbSet<Post> Posts { get; set; }
}
```

Note that enabling "Multiple Active Result Sets" is recommended. This allows two queries to be active at the same time. For example, this would be required to query for posts associated with a blog while enumerating all blogs.

**Database Initializers** By default, Code First will create a database automatically if the database it targets doesn't exist. For some folks, this will be the desired functionality even when deploying, and just the production database will be created the first time the application launches. If you have a DBA taking care of your production environment, it's far more likely that the DBA will create the production database for you, and once your application is deployed it should fail if the database it targets doesn't exist. In this article, I've also overridden the default initializer logic and have configured the database to be dropped and recreated whenever my schema changes. This is definitely not something you want to leave in place once you deploy to production.

The recommended approach for changing or disabling initializer behavior when deploying is to use the App.config file (or Web.config for Web applications). In the appSettings section, add an entry whose key is DatabaseInitializerForType, followed by the context type name and the assembly in which it's defined. The value can either be "Disabled" or the initializer type name followed by the assembly in which it's defined.

The following example disables any initializer logic for the context I've been using in this article:

```
<appSettings>
  <add
    key="DatabaseInitializerForType Blogging.BlogContext, Blogging"
    value="Disabled" />
</appSettings>
```

The following example will change the initializer back to the default functionality that will create the database only if it doesn't exist:

```
<appSettings>  <add
    key="DatabaseInitializerForType Blogging.BlogContext, Blogging"
    value="System.Data.Entity.CreateDatabaseIfNotExists EntityFramework" />
</appSettings>
```

**User Accounts** If you decide to let your production application create the database, the application will need to initially execute using an account that has permissions to create the database and modify schema. If these permissions are left in place, the potential impact of a security compromise of your application is greatly increased. I highly recommend that an application is run with the minimal set of permissions required to query and persist data.

> By default, Code First will create a database automatically if the database it targets doesn't exist.

## More to Learn

Summing up, in this article, I took a quick look at getting started with Code First development and the new DbContext API, both of which are included in the ADO.NET Entity Framework 4.1. You saw how the fluent API can be used to map to an existing database or to affect the shape of a database schema that's generated by Code First. I then looked at the change tracker API and how it can be used to query local entity instances and additional information about those instances. Finally, I covered some considerations for deploying an application that uses Code First for data access.

If you'd like to know more about any of the features included in the Entity Framework 4.1, visit msdn.com/data/ef. You can also use the Data Developer Center forum to get help with using the Entity Framework 4.1: bit.ly/166o1Z. ■

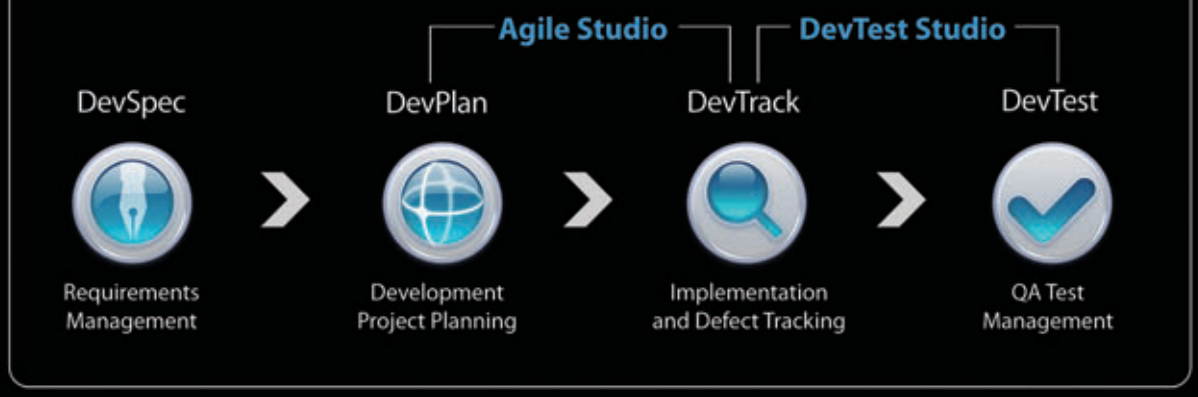
**ROWAN MILLER** *is a program manager on the Entity Framework team at Microsoft. You can learn more about the Entity Framework on his blog at romiller.com.*

**THANKS** to the following technical expert for reviewing this article: *Arthur Vickers*

# Gain Full Traceability

## Requirements Driven Quality Management

### DevSuite




## A Modular Approach to ALM

- DevSuite can be purchased and deployed as a complete ALM solution or as individual modules
- Convenient bundles provide solutions for quality management (DevTest Studio) and Agile development (Agile Studio)
- Individual modules and bundles can be easily expanded when you need additional functionality

## Agile Ready

- DevSuite is a hybrid platform that allows you to mix elements from multiple methods, including traditional development, to achieve the right balance for your team
- Out-of-the box methodology templates for Scrum, XP, Test Driven, Waterfall, and Iterative development


Download your FREE 30-day trial of DevSuite now at: www.techexcel.com/downloads

www.techexcel.com

1-800-439-7782

# Consuming External OData Feeds with SharePoint BCS

Eric White

Microsoft Business Connectivity Services (BCS) is a feature of Microsoft Office 2010 and SharePoint 2010 that helps developers and users bring data into SharePoint. Surfacing external data in SharePoint enables users to build composite applications that give them better access to critical information and make their interactions with that information more convenient.

BCS provides three basic mechanisms that you can use to bring external data into SharePoint. First, you can connect to and consume databases via SQL queries. By default, SQL Server is supported. With some work, you can connect to MySQL, Oracle and other database-management systems.

Second, you can consume Web services that expose methods that follow specific patterns for the method prototypes.

Third, you can use the Microsoft .NET Framework and C# or Visual Basic code to connect to data sources. The most common approach is to write a .NET Assembly Connector.

This article discusses:

- OData and SharePoint
- CRUD operations with OData
- Building a .NET Assembly Connector
- Viewing external lists in SharePoint

Technologies discussed:

SharePoint 2010 BCS, OData

In this article, I'll show you the third approach: writing a .NET Assembly Connector that consumes an Open Data Protocol (OData) feed.

## Why a .NET Assembly Connector?

With the burgeoning proliferation of OData feeds, you may need to consume one of those feeds to enable interesting functionality for your users. BCS doesn't have a built-in capability to consume OData feeds, but it's relatively easy to write a .NET Assembly Connector to do so.

This is also a convenient approach for demonstrating how to write a .NET Assembly Connector. Another approach would be to write a custom connector to consume a SQL database, but this is superfluous because BCS can easily consume SQL data sources out of the box. Further, to demonstrate writing a .NET Assembly Connector that consumes a database requires that you install and configure the database appropriately. This is not difficult, but it adds some extra steps and complicates the example. In contrast, writing a .NET Assembly Connector that consumes an existing OData feed could not be simpler.

This example also shows you how to implement Create, Read, Update and Delete (CRUD) operations using OData. You'll see just how easy this is.

You'll probably be surprised to see just how little code you need to write in order to create a .NET Assembly Connector that consumes an OData feed. Data-access technologies have come a long

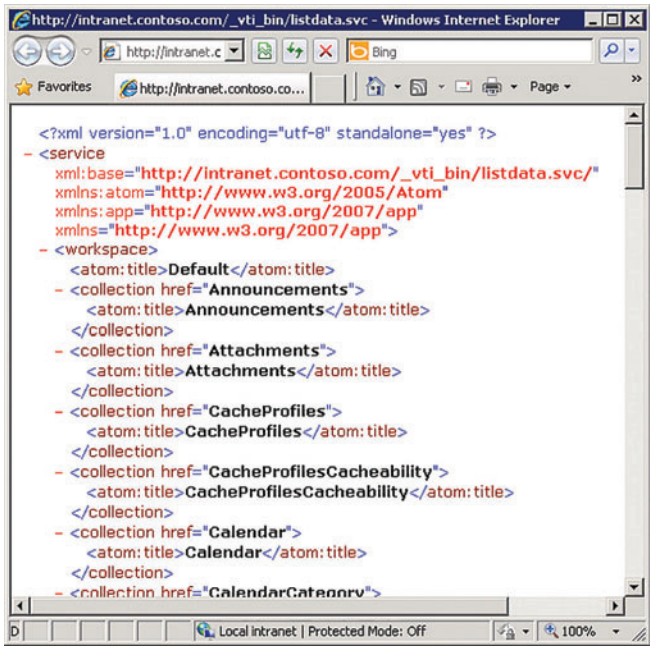


Figure 1 **OData from a SharePoint List**

way, and OData promises to enable a new level of interoperability between applications that produce and consume data.

Note that SharePoint designer is another approach to model, develop and publish BCS external context types. SharePoint Designer natively supports building business entity models from back-end SQL databases and Web services with relatively flat data structure. Use of SharePoint Designer simplifies (and reduces) BCS development work. However, it does not natively support OData services currently.

## OData and SharePoint

OData is a Web protocol for querying and updating data that builds upon existing Web technologies such as HTTP, Atom Publishing Protocol (AtomPub) and JSON. OData is being used to expose and access information from a variety of sources including relational databases, file systems, content management systems and traditional Web sites. For a good introduction to OData, see "Building Rich Internet Apps with the Open Data Protocol" (msdn.microsoft.com/magazine/ff714561), from the June 2010 issue of *MSDN Magazine*. The article was written by Shane Burgess, one of the program managers in the Data and Modeling Group at Microsoft.

SharePoint Foundation 2010 and SharePoint Server 2010 expose list data as an OData feed. This functionality is enabled by default. If you have a SharePoint site installed at the URL http://intranet.contoso.com, you can retrieve the set of SharePoint lists for the site by entering http://intranet.contoso.com/_vti_bin/listdata.svc into your browser.

If your corporate SharePoint 2010 site includes the My Site feature and your alias is, say, ericwhite, you can see the lists exposed on your My Site by entering http://my/sites/ericwhite/_vti_bin/listdata.svc into your browser. In either case, you'll see an atom feed like the one in **Figure 1** displayed in the browser.

With the .NET Framework, a simple way to consume an OData feed is to use WCF Data Services. You use Visual Studio to add a service reference to the OData feed, and the IDE automatically generates code so you can use strongly typed classes to query and update the feed. I'll walk you through this process.

For more information about how this works, see the WCF Data Services Developer Center (msdn.microsoft.com/data/bb931106), which has a beginner's guide and links to resources.

## Getting Started

As I mentioned earlier, SharePoint 2010 exposes list data as OData feeds. An easy way to access the feeds is through a .NET Assembly Connector, and I'm going to walk you through the process of building that connector.

This process will create an external content type that you can display and maintain in an external list. This might seem a little bit funny—after you have the example working, you'll have a SharePoint site that contains two lists with exactly the same data. One of the lists will be a SharePoint list that you create and set up in the usual way. The other list will be an external list that displays the data coming from the OData feed for the first list. If you add or alter records in one of the lists, the changes show up in the other one.

The primary benefit of this approach is that it's simple to build and run the example. You don't need to install any infrastructure for the example. All you need is a SharePoint farm for which you have farm administrator access.

> OData is a Web protocol for querying and updating data.

If you don't already have the infrastructure, the easiest way to run the example is to download the 2010 Information Worker Demonstration and Evaluation Virtual Machine (bit.ly/gBKog8). The virtual machine, or VM, comes complete with an installed, working copy of SharePoint 2010, Visual Studio 2010, Office 2010 and much, much more. The example demonstrated in this article works without modifications in this VM.

If you have your own SharePoint 2010 development environment, it's easy to modify this example appropriately (I'll indicate where as I go along). However, if you're just getting started with SharePoint development and want to try out a few examples, the VM is the way to go.

The first step in building this example is to become familiar with using OData to manipulate data in a SharePoint list. In this example, you're connecting to a customer relationship management (CRM) system that exposes a customer list using OData.

First, create a SharePoint list that contains a few records that represent customers.

1. Open a browser, browse to a SharePoint site, and create a custom list named Customers.
2. Change the name of the Title column to CustomerID.

Figure 2 **Updated Code for Program.cs**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Net;
using Contoso.Crm;

class Program {
  static void Main(string[] args) {
    TeamSiteDataContext dc =
      new TeamSiteDataContext(new Uri(
      "http://intranet.contoso.com/_vti_bin/listdata.svc"));
    dc.Credentials = CredentialCache.DefaultNetworkCredentials;
    var customers =
      from c in dc.Customers
      select new {
        CustomerID = c.CustomerID,
        CustomerName = c.CustomerName,
        Age = c.Age,
      };
    foreach (var c in customers)
      Console.WriteLine(c);
  }
}
```

3. Create a new column named CustomerName, with a type of Single line of text.
4. Create a new column named Age, with a type of Number.
5. Add a few records.

Now, log into the VM as administrator and start Visual Studio 2010. Create a new Windows Console Application. For building these OData samples, it doesn't matter whether you build for .NET Framework 4 or .NET Framework 3.5. However, when building the .NET Assembly Connector later, you'll need to target .NET Framework 3.5 because it's the only version currently supported by SharePoint 2010. To have Visual Studio generate classes with good namespace names, name this project Contoso.

Later in this article, I'll discuss namespace names for both OData and BCS .NET Assembly Connectors. There are specific things you can do to generate namespace names properly, and in this case, the namespace name will make the most sense if you name the project Contoso.

On the Visual Studio menu, click Project, then click Add Service Reference. Enter the OData service URL of the SharePoint site in the Add Service Reference dialog box. If you're using the demo VM, the service URL is http://intranet.contoso.com/_vti_bin/listdata.svc.

If you're connecting to a SharePoint site at a different URL, you'll need to adjust the service URL as appropriate.

Click Go. Visual Studio will attempt to go to the location and download metadata from the SharePoint site. If successful, it will display the service name in the Services list in the Add Service Reference dialog box. Because you're simulating a CRM system, enter Crm into the Namespace field. Click OK. It's interesting to examine the generated code. Click the Show All Files button in the Solution Explorer window, then expand the Crm namespace, expand Reference.datasvcmap and open Reference.cs. It'll look something like this (comments removed for clarity):

```
namespace Contoso.Crm {
  public partial class TeamSiteDataContext :
    global::System.Data.Services.Client.DataServiceContext {
    ...
```

Because of how you named the project and the service reference namespace, the namespace for the generated classes is Contoso.Crm. The fully qualified name of the class for the Customers list is Contoso.Crm.Customers, which makes sense.

Also note the generated name for the data context. In this case, it's TeamSiteDataContext. The generated name of this class is based on the name of the SharePoint site that you connect to. In the case of the demo VM, the name of the default site that you connect to is Team Site. If your environment is different, note the name of the data context class so that you can alter code in examples appropriately.

Open Program.cs and update it with the code shown in **Figure 2**. If you're not using the demo VM, adjust the OData service URL accordingly. Compile and run the program to see the results of the query. As you can see, it doesn't take a lot of code to retrieve data from a list using OData.

## CRUD Operations with OData

Now let's try inserting a customer. Your code will need to create a new CustomersItem and initialize its values. Then it calls the TeamSiteDataContext.AddToCustomers method, passing the CustomersItem object as a parameter. Finally, to commit changes you'll call the TeamSiteDataContext.SaveChanges.

In Program.cs, replace the customers variable declaration and foreach loop with the following code:

```
CustomersItem cust = new CustomersItem();
cust.CustomerID = "JANE08";
cust.CustomerName = "Jane";
cust.Age = 22;
dc.AddToCustomers(cust);
dc.SaveChanges();
```

To update a customer, you'll query the Customers list, retrieving the specific customer to update. Then update properties as appropriate. Call the TeamSiteDataContext.UpdateObject method. Finally, call the TeamSiteDataContext.SaveChanges to commit changes:

```
CustomersItem cust = dc.Customers
  .Where(c => c.CustomerID == "BOB01")
  .FirstOrDefault();
if (cust != null) {
  cust.CustomerName = "Robert";
  dc.UpdateObject(cust);
  dc.SaveChanges();
}
```

To delete a customer, query the Customers list, retrieving the specific customer to delete. Call the TeamSiteDataContext.DeleteObject method. Call the TeamSiteDataContext.SaveChanges to commit changes:

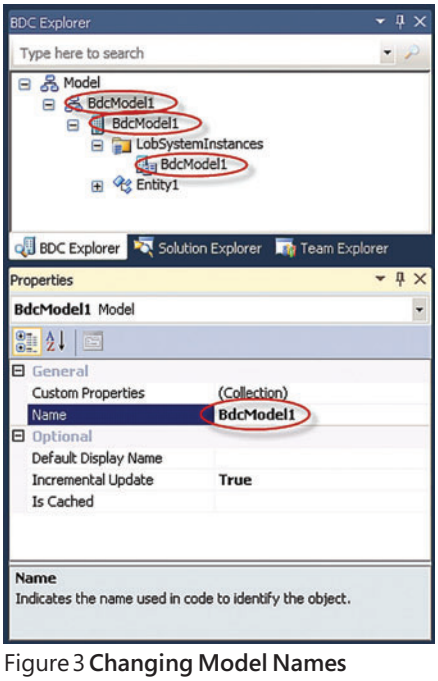



Figure 3 **Changing Model Names**

# TEAM FOUNDATION SERVER HOSTING IS HERE!

ONLY $20/mo per user

## LIMITED TIME SPECIAL OFFER: FIRST 30 DAYS FREE & NO SETUP FEES

Instead of spending your valuable time and resources maintaining a TFS server environment, you have a NEW option - to use the DiscountASP.NET Hosted TFS SaaS solution The DiscountASP.NET Hosted TFS service is a SaaS solution for source code version control and bug tracking. Start saving money and time so you can focus on what you do best – developing killer software!

## SIMPLIFY YOUR LIFE CYCLE TODAY!

### TFS HOSTING FFEATURES:

- Source Code Version Control
- Bug/Issue Tracking
- Web Team Access
- Unlimited Projects
- 5gb of Disk Space
- Visual Studio 2010 Integration
- Visual Studio 2008 Integration
- Web-Based Control Panel
- Month-to-month Billing

***New TFS Add-on:***
*Urban Turtle – scrum tool for agile project management.*

## FOR MORE INFO: www.DiscountASP.NET/tfs/simplify

discountASP.net
*Team Foundation Server Hosting*

DevPro 2010 Community Choice GOLD

Members Choice Winner The Code Project 2010

2010 Readers Merit Visual Studio Magazine

Microsoft GOLD CERTIFIED Partner

```
CustomersItem cust = dc.Customers
  .Where(c => c.CustomerID == "BILLO2")
  .FirstOrDefault();
if (cust != null) {
  dc.DeleteObject(cust);
  dc.SaveChanges();
}
```

As you can see, altering a SharePoint list using OData is simple.

## Building a .NET Assembly Connector

In the process of building the .NET Assembly Connector, you define a class that represents the entity that you'll expose as an external content type. In this example, you define a Customer class that represents an item in the Customers list. Some methods, such as the method to create a customer, take an instance of this class as an argument. Other methods, such as the method to return all customers in the list, return a collection of instances of this class.

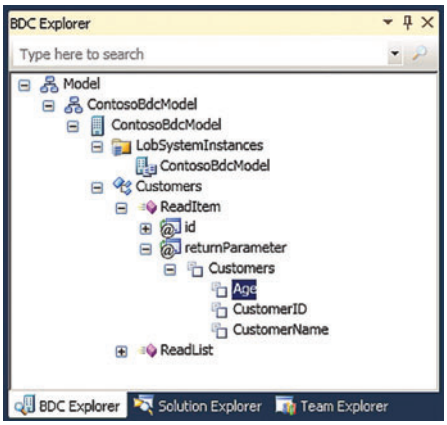



Figure 4 **The Completed Customers Entity**

You'll also configure a BCS model that describes this class in an XML dialect. The infrastructure underlying BCS will use the information in the XML definition of the BCS model so that the external content type is usable from within SharePoint 2010.

If there's one key point you should take away from this article, it's this: You must make the model match the actual defined class exactly.

There are tools that help you make the BCS model definition match the actual class. However, the key point is that through one approach or another, you need to carefully validate that the model matches the actual class.

In a more typical BCS implementation, you'll define many of these classes and model all of them in the BCS model definition. The bulk of the work when implementing a complex BCS solution is to make the model match the classes.

Now let's build the connector. For this example you'll just build a read-only .NET Assembly Connector, but once you've seen the basics, it should be straightforward to add the rest of the CRUD functionality.

The code download for this article includes the code and the BCS model for CRUD functionality; it works without modification in the demo VM.

Log into your SharePoint development computer as administrator. You must have farm administrator rights to build and deploy a .NET Assembly Connector.

Start Visual Studio 2010. Create a new project. Create a new SharePoint 2010 Business Data Connectivity (BDC) Model application. As before, you must target the .NET Framework 3.5. Name the project Contoso and click OK. Again, we'll use the project name Contoso, which will be used in the namespace.

In the SharePoint Customization Wizard you can enter a local site URL of a SharePoint site for debugging. In this wizard on the demo VM, the URL is correctly set by default to http://intranet.contoso.com. Change this URL if you're working with a SharePoint site at a different address. The wizard also lets you know that this project will be deployed as a farm solution. Click Finish. Wait a bit for the wizard to run.

Rename the BDC model nodes in the Solution Explorer from BdcModel1 to ContosoBdcModel.

Next, open the BDC Explorer pane (by default right next to the Solution Explorer). Rename the three BdcModel1 nodes to ContosoBdcModel. In the BDC Explorer, you can't directly rename each node in the tree control. Instead, you need to select each node and then modify the name in the Properties pane (see **Figure 3**).

The next step is to rename the entity and specify the identifier for the entity. Select Entity1 in the BDC Designer. After selecting the entity, you can change its name in the Properties pane. Change its name to Customers, and change its namespace to Contoso.Crm.

In the BDC Designer, click on Identifier1, and change its name to CustomerID. You also need to design the entity in the BDC Explorer. This part needs to be done precisely. If there's a mismatch between the BDC model and the actual class that you'll be using, the results are undefined, and error messages are not always illuminating. In some cases, your only clue to what is wrong is that the list Web Part for the external content type won't load.

> You must make the model match the actual defined class exactly.

Expand the nodes in the BDC Explorer until you can see the Identifier1 node under the id parameter of the ReadItem method. Change its name to CustomerID. Expand the tree until you can see the Entity1 node under the returParameter for the ReadItem method. Change the name of the entity to Customers, and change the type name to Contoso.Crm.Customers, ContosoBdcModel.

You're going to completely redefine the Customer entity, so delete the Identifier1 and Message nodes from the Customers entity.

Right-click on Customers and click Add Type Descriptor. Rename the name of the new type descriptor to CustomerID. By default, the type of a new type descriptor is set to System.String, which is what you want for this example. In the Properties pane, scroll down until you see the Identifier field. Use the drop-down list to change its value to CustomerID.

Again, right-click on Customers and click Add Type Descriptor. Rename to CustomerName. Its type is System.String, which is correct.

Add another type descriptor, rename to Age, and change its type to System.Int32. After making these changes, the BDC Explorer pane will look like **Figure 4**. Expand the ReadList node, expand the returnParameter node and rename Entity1List to CustomersList. The type name is set to the following:

```
System.Collections.Generic.IEnumerable`1[[Contoso.BdcModel1.Entity1,
ContosoBdcModel]]
```

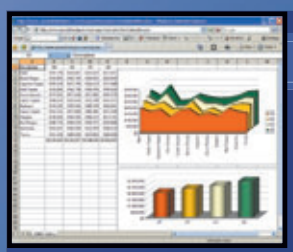
This syntax, consisting of the back tick followed by the one (`1) is the syntax that represents a generic class with one type parameter.

# In a League by Itself

*"SpreadsheetGear 2010 is Fantastic! These new capabilities just propelled this control way-way past any competition, of which you have none IMHO. SpreadsheetGear is in a league all by itself."*

Greg Newman, Senior Software Engineer, WSFS Cash Connect

# SpreadsheetGear 2010



## ASP.NET Excel Reporting

Easily create richly formatted Excel reports without Excel using the new generation of spreadsheet technology built from the ground up for scalability and reliability.

## Excel Compatible Windows Forms Control

Add powerful Excel compatible viewing, editing, formatting, calculating, charting and printing to your Windows Forms applications with the easy to use WorkbookView control.

## Create Dashboards from Excel Charts and Ranges

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

**Download the FREE fully functional 30-Day evaluation of SpreadsheetGear 2010 today at**

**www.SpreadsheetGear.com.**

SpreadsheetGear

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

The type that follows in the double square brackets consists of a fully qualified type, as well as the model name of the BDC model that contains the type. Change the type name to:

```
System.Collections.Generic.IEnumerable`1[[Contoso.Crm.Customer,
ContosoBdcModel]]
```

This corresponds to a type of IEnumerable<Contoso.Crm.Customer>, where the Customer type is found in the ContosoBdcModel.

Delete the Entity1 node that's a child of the CustomersList node. Copy the Customers entity that you recently configured as a child type descriptor of the returnParameter of the ReadItem method.

Select the CustomersList node that's under the returnParameter of the ReadList method, and paste the Customers entity.

Return to the Solution Explorer window, and edit the Feature1.Template.xml file. Add a SiteUrl property with a value of the URL of the SharePoint site:

```
<?xml version="1.0" encoding="utf-8" ?>
<Feature xmlns="http://schemas.microsoft.com/sharepoint/">
  <Properties>
    <Property Key="GloballyAvailable" Value="true" />
    <Property Key="SiteUrl" Value="http://intranet.contoso.com" />
  </Properties>
</Feature>
```

Rename Entity1.cs to Customers.cs. Replace the contents of Customers.cs with the following code, which defines the Customer entity for the assembly connector:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Contoso.Crm {
  public partial class Customer {
    public string CustomerID { get; set; }
    public string CustomerName { get; set; }
    public int Age { get; set; }
  }
}
```

Replace the code in CustomersService.cs with the code in **Figure 5**, which defines the methods that retrieve a single item and retrieve a collection of items.

Following the procedures that I described at the beginning of this article, add a service reference to the site, specifying a namespace of Crm. Click OK. Build the solution by clicking Build | Build Contoso. Now deploy the solution by clicking Build | Deploy Contoso.

Give permissions for all authenticated users to access this external content type. Open Central Administration, then click Manage Service Applications. Click Business Data Connectivity Service. Click on the down arrow next to the external content type that you just added, and then click Set Permissions. Click the Browse button just below the text box. Click All Users, then click All Authenticated Users. Click the Add button, then click OK.

Back in the Set Object Permissions window, click the Add button, and then set permissions for Edit, Execute, Selectable In Clients and Set Permissions. Click OK.

Browse to the SharePoint site. Create a new External List. Specify CustomersExt for the name of the list. Click on the external content type browser. In the External Content Type Picker, click on the external content type you just created. If you're using the demo VM, the content type you just created will be the only external content type in the list. Click OK.

Click the Create button, wait a bit and, if all goes well, you'll see a new external list that contains the same data as the Customers list.

Figure 5 **CustomersService.cs**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Text;
using Contoso.Crm;

namespace Contoso.Crm {
  public class CustomersService {
    public static Customer ReadItem(string id) {
      TeamSiteDataContext dc = new TeamSiteDataContext(
        new Uri("http://intranet.contoso.com/_vti_bin/listdata.svc"));
      dc.Credentials = CredentialCache.DefaultNetworkCredentials;
      var customers =
        from c in dc.Customers
        where c.CustomerID == id
        select new Customer() {
          CustomerID = c.CustomerID,
          CustomerName = c.CustomerName,
          Age = (int)c.Age,
        };
      return customers.FirstOrDefault();
    }

    public static IEnumerable<Customer> ReadList() {
      TeamSiteDataContext dc = new TeamSiteDataContext(
        new Uri("http://intranet.contoso.com/_vti_bin/listdata.svc"));
      dc.Credentials = CredentialCache.DefaultNetworkCredentials;
      var customers =
        from c in dc.Customers
        select new Customer() {
          CustomerID = c.CustomerID,
          CustomerName = c.CustomerName,
          Age = (int)c.Age,
        };
      return customers;
    }
  }
}
```

If you add a record to the regular list, you'll see it show up in the external list. You can play around with the data a bit. Add, delete or change some records in the regular SharePoint list and see the data show up in the external list.

## Pulling the Pieces Together

As you've seen, it's pretty easy to use WCF Data Services to query and modify an OData feed. It's also a straightforward process to create an external content type via a .NET Assembly Connector. You can connect to just about any data source using a .NET Assembly Connector.

This was just a simple example employing a readily accessible data source, but the pattern is easily extended to any data source with an OData feed. To learn more about BCS customization, see the MSDN Library page "Business Connectivity Services How-tos and Walkthroughs" (msdn.microsoft.com/library/ee557949). There's also a trove of information about OData at the MSDN Data Developer Center (msdn.microsoft.com/data/bb931106) and the Open Data Protocol site (odata.org). ■

**ERIC WHITE** *is an independent software developer and author with 25 years of experience developing enterprise applications on a variety of platforms. He currently specializes in Microsoft Office technologies, including Open XML, SharePoint and Office client development. You can follow him on Twitter at twitter.com/EricWhiteDev or on his blog at ericwhite.com/blog.*

**THANKS** to the following technical experts for reviewing this article: *Ying Xiong*

JAIME RODRIGUEZ

# Windows Phone 7 Tombstoning

A good mobile platform should acknowledge the hardware constraints that mobility imposes on the device. Compared to desktops, mobile devices have less memory, less processing power, limited screen real-estate and limited battery life. Add up these constraints and you should conclude that, on a non-dedicated device where many applications will run, applications will eventually be closed or shut down to make resources available to other applications.

Windows Phone deals with this constraint through a feature called tombstoning. While it may seem like tombstoning would be a straightforward proposition, it's actually a point of contention among developers. Some argue it shouldn't be needed. Others argue that it's too difficult. The rest of us simply hate the name of the feature. Still, mobile device constraints make it a necessary evil, so great mobile applications must be able to handle tombstoning.

## Windows Phone Application Lifecycle

Most new Windows Phone developers come to the platform with the expectation that an application's lifecycle looks something like:

1. Start
2. Run
3. Exit
4. Go back to 1 and start again

Windows Phone 7 challenges this expectation by exposing a different lifecycle—one that's less process-oriented and more session-oriented.

In Windows Phone 7 you should think of the lifecycle as:

1. Start
2. Run
3. Interrupted execution or exit
4. If interrupted, come back—or, even if interrupted, start anew
5. If exited, start anew

The benefit of this new session-oriented model is that users can navigate across apps without having to think about how the OS is managing its resources. A user doesn't care if interrupting their game to reply to an incoming SMS message will kill the process for the game. The user should expect that he can get back to the game when finished with the message. If that works well enough, the underlying details are irrelevant.

The downside for developers is that there's a bit more to handle to truly provide the perception of session continuity because your sessions are still running on a traditional, process-centric OS. To accommodate sessions in a process-centric world, you create logical states for the session: Launched, Activated, Running, Deactivated, Tombstoned and Closed (or Ended).

Code download available at code.msdn.microsoft.com/mag201105MobileMatters.

**Figure 1** shows the practical lifecycle for a Windows Phone 7 app. The application lifecycle events, described in **Figure 2**, are exposed by the Microsoft.Phone.Shell.PhoneApplicationService class.

The tombstoned state is a bit more complicated and not directly related to a PhoneApplicationService event. When an application is deactivated, the OS does not immediately kill the process for that application. In theory, the OS kills the application when it needs resources and when it happens. The application doesn't get a notification at all, and it's simply killed.

In practice, Windows Phone 7 kills the process quickly after control is transferred to another foreground application, but this is not a detail you should count on. Microsoft already announced at Mobile World Congress last February that improvements such as Fast App Switching are coming, so don't rely on an implementation detail to determine when tombstoning happens. Instead, prepare for it by doing the right work at deactivation.

## Deactivated vs. Tombstoned

A phone has many processes running all the time (shell, phone and so on), but it has, at most, one application running in the foreground. (It can have zero when nothing is running in the foreground.)

When a foreground application transfers control to another application or to an OS component, it gets deactivated. After a process is deactivated, the OS might kill the process to release the resources. This is called tombstoning.

As you can see, tombstoning does not happen every time the app is deactivated, but tombstoning always comes after a deactivation. In fact, Deactivated is the last event PhoneApplicationService fires before tombstoning, so this is where you must do your work to activate the app again later.

**Figure 3** shows all the different tasks that lead to deactivation, and guesses the likelihood of tombstoning occuring.

There's a subset of choosers that don't tombstone immediately, but can still be tombstoned if the user takes an action that tombstones the process. These include PhotoChooserTask (unless the user specifies crop), CameraCaptureTask, MediaPlayerLauncher, EmailAddressChooserTask and PhoneNumberChooserTask.

All other choosers and launchers tombstone right after the Show method is called.

To see the Windows Phone 7 application lifecycle in action, launch the LWP.TombStoning sample in the code download.

## Saving and Restoring State

Because the goal for session-based navigation is to make it easy for the user to jump across foreground applications seamlessly, you must save

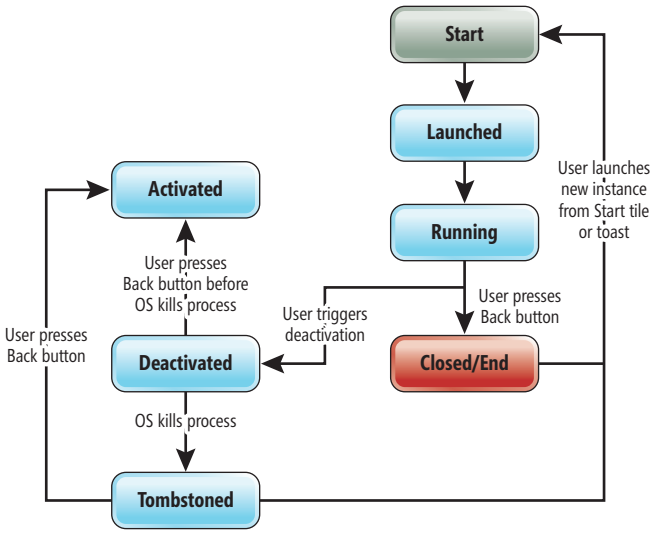



Figure 1 **Windows Phone 7 Application Lifecycle**

all relevant state in the Deactivated event and restore state in the Activated event. Most applications have three types of state to manage:

- **Persistent application state** must always be persisted. This includes application settings, user data and so on.
- **Session-specific application state** includes temporary state such as caches and ViewModels that need to be restored in activation, but are started anew in a fresh launch of the app.
- **UI- or page-specific state** is needed to restore a PhoneApplicationPage when an app is activated. Windows Phone 7 saves the back stack for an application when tombstoning. Upon activation it restores only the last active page before the application was tombstoned. If the user presses the Back button, the previous page gets instantiated.

Persistent application state should be saved in IsolatedStorage or via the ApplicationSettings class. Application state should be saved as early as possible, in case the battery runs out, for example.

If it's user data (such as a session cache) and you don't want to serialize too often, then it should be saved both on the Deactivated and Closing events, and (symmetrically) it should be restored in the Activated or Launching events.

Session-specific state can be saved in isolated storage if you want control over serialization formats or you have too much data, or it can be saved in the PhoneApplicationService.State dictionary. You should save it only in the Deactivated event and restore it in the Activated event.

Page-specific state should be saved in the PhoneApplicationPage.State dictionary. The key to saving page state is to remember that the application has a back stack of pages that will be serialized automatically when PhoneApplicationService.Deactivated occurs. To keep your pages ready for tombstoning, you must listen for the PhoneApplicationPage.OnNavigatedFrom override in your page and save any view state that's not yet committed to your Model (or ViewModel) in the page's dictionary. Don't wait until the Deactivated event, because by then you can't get to the pages in the back stack.

Of course, if you save page state when you recieve OnNavigatedFrom, you should restore it in the OnNavigatedTo override for the page.

You could also save page-specific state in ViewModels, and then serialize your ViewModels as session state, but that would require saving uncommitted state on a ViewModel, so I don't recommend it. Leverage the infrastructure in place to stay future-proof for later optimizations in the platform.

## Avoiding Pitfalls

Tombstoning is not difficult. It's just a bit of tedious work and it demands consistency and planning. If your app is deactivated but not tombstoned, your state will remain in memory and will not be reconstructed.

Avoid relying on class constructors that create state that your application needs, but that might be released during deactivation. Symmetry is preferred. Use the PhoneApplicationService Deactivated and Activated events for app-level state, and OnNavigatedFrom or OnNavigatedTo for page state.

If you have objects (singletons) in your app that are instantiated outside of activation calls (maybe due to delay instantiation), always check whether they're properly constructed and initialized before trying to use them. A common mistake that I've encountered is reading data in the PhoneApplicationService.Activated event or PhoneApplicationPage.OnNavigatedTo and not resetting it. Pages can be NavigatedTo multiple times (regardless of tombstoning) and even a session can be tombstoned multiple times during a session.

After you've restored state, clear it. You can set it later in the NavigatedFrom override for your page or the Deactivated event for the app.

Be smart about what you save and when you restore it. Part of making it seamless for the user to come back to the app is restoring

Figure 2 **Application Lifecycle Events**

| Logical State | PhoneApplicationService Event | Description |
|---|---|---|
| Launched | Launching | Application is launched when the user presses the start tile or the application list icon, or the user clicks on a toast notification. Launched is a fresh start for the session. |
| Activated | Activated | Application is activated when the user presses the Back button and brings the application that had previously been deactivated back to the foreground. In this case, the user expects to be coming back to an ongoing session. |
| Running | Running | After being either launched or activated, the application is running. |
| Deactivated | Deactivated | An application that's running is deactivated when foreground processing is transferred from this application to another application or to an OS component (such as a Chooser or a Launcher or the lock screen). The session is interrupted, but is expected to be resumed later. |
| Ended (or Exited) | Closing | User is exiting the application by pressing the back key on the main page. When exiting, the user expects to come back to a new, fresh application. |

Figure 3 **Deactivation Tasks**

| Action | Deactivated? | Tombstoned? |
|---|---|---|
| User presses the Back button on the first page of the application | No; this closes the app | No. Deactivation never happened. |
| User presses the Start button | Yes | Very likely, but not guaranteed. A few seconds after the app is deactivated, it's tombstoned. If the user comes back to the app quickly after deactivation, it might not tombstone. This could be considered an indeterminate timeout. |
| User invokes a chooser or a launcher that tombstones | Yes | Very likely, but timeout applies. |
| User invokes a chooser or a launcher that doesn't tombstone | Yes | Less likely, but it can still happen. If the user presses the Start button in the middle of the task, then sees the "User presses the Start button" rule. A new Deactivated event is not fired because the app was already deactivated. |
| Lock screen comes up and app is not configured to run under lock | Yes | Very likely, but timeout applies. |
| A toast notification comes in and user taps on it, transferring to another foreground application | Yes | Very likely, but timeout applies. |

your application promptly. If you save too much of either page or application state, it will slow down activation. Leverage isolated storage if needed to background load state that might not be required immediately when your application is activated. Both activation and deactivation should happen in less than 10 seconds. Be aware of this or the OS may kill your process before it finishes deactivating or as it's reactivated. You should, however, aim for much less than 10 seconds.

Understand the constraints of the serialization framework. You can, at most, store around 2MB across all of your page and application state. If your total adds up to more, you'll start seeing exceptions when you navigate and when you deactivate. You shouldn't be serializing this much data in page state. If you need to cache large data sets, keep them in isolated storage.

Use query strings for page navigation. If you must pass context into a new page, use the query string passed to the page to either pass all the data or pass a unique identifier (a token) to a service locator that can fetch the data based on that token. Don't assume ViewModels or page state are available when your pages are activated after tombstoning.

Understand your choosers and launchers. Not all of them tombstone, and there are specific rules on how to wire up event listeners for choosers. For these rules, please read "How to: Use Choosers for Windows Phone" at bit.ly/edEsGQ.

Be mindful of the relationship between OnNavigatedTo and PhoneApplicationPage.Loaded. Within OnNavigatedTo, the visual tree for the page is not fully built yet. Often you'll have to extract restored state, but wait until the page's Loaded event to restore the UI state. Examples of actions that must be delayed include setting Focus, setting SelectedIndex in a pivot and scrolling.

If you're doing a lot to save and restore data—and you shouldn't be—consider some advanced optimizations. Note that you should do this only if necessary and make sure to test thoroughly.

To detect tombstones, set a flag on your app class in Deactivated. If the flag isn't reset on Activated, it means you weren't tombstoned and all your pages should still be in memory and require no restoration. To combine it with detection for tombstoning in pages, you can use a token for each activation within a session.

Another optimization is to listen for the page's OnNavigatingFrom override and detect direction. If NavigationMode is going back, the page will be destroyed and there's no reason to save state for it.

Again, planning is the key to proper tombstoning. Don't leave tombstoning for the end of the app development cycle and try to retrofit it. Plan it early, implement it as you go and test thoroughly.

## You Can Make It Great

One last tip for developers is to think hard about making the experience seamless. Windows Phone 7 controls allow you to get around a page easily. To make it truly seamless for the user—to feel like he never left a page or app—you should consider restoring the following:

- SelectedIndex for the pivot (if the page contains a pivot)
- Scrolling position in a ListBox or any other ScrollViewer in the page
- Zoom levels and other transforms in a map control, picture viewer or any other control that supports manipulation
- Uncommitted text in a TextBox; if the TextBox is critical to the app (the tweet text in a Twitter app, for example) consider restoring text selection, caret position and focus
- Element that had focus in a page (especially if it's a TextBox, which needs to show SIP)

The one you shouldn't restore is SelectedItem in a panorama. The panorama doesn't support this, and setting the DefaultItem in a panorama is not the same as panning to the correct page. I recommend that you avoid using DefaultItem as a means to get back to the panorama item selected before tombstoning.

To see these tips in action, launch the LPW.TombstoningWithState sample in the code download. The readme.txt file has pointers and scripts for each scenario.

## A Few Last Remarks

This article is by no means a comprehensive reference to tombstoning. But now that you know what to look for, start introducing some tombstoning patterns into your own Windows Phone 7 apps. I think you'll see right away how much it improves the user experience. ■

**JAIME RODRIGUEZ** *is a principal evangelist at Microsoft driving adoption of emerging client technologies such as Silverlight and Windows Phone 7. You can reach him on Twitter at twitter.com/jaimerodriguez or read his blog at blogs.msdn.com/jaimer.*

**THANKS** to the following technical expert for reviewing this article: *Peter Torr*

BEST SELLER

## TX Text Control .NET for Windows Forms/WPF | from $564.27

TX TEXT CONTROL word processing components

**Word processing components for Visual Studio .NET.**

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

BEST SELLER

## FusionCharts | from $189.05

InfoSoft Global
empowering human thoughts

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

- Liven up your web applications using animated & data-driven charts
- Create AJAX-enabled charts with drill-down capabilities in minutes
- Export charts as images/PDF and data as CSV from charts itself
- Create gauges, dashboards, financial charts and over 550 maps
- Trusted by over 17,000 customers and 330,000 users in 110 countries

BEST SELLER

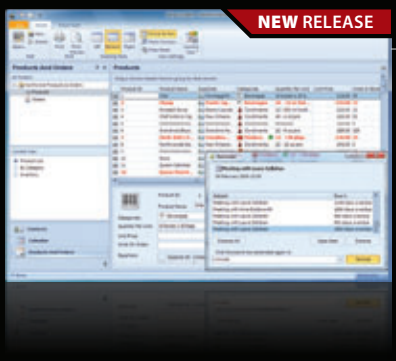
## ActiveReports 6 | from $559.20

GrapeCity PowerTools

**Latest release of the best selling royalty free .NET report writer.**

- Now supports Windows Server 2008, 64Bit & IE8.0
- First Flash based Report Viewer for end users
- Supports PDF Digital Signatures, RSS Bar Codes and external stylesheets
- Features an easy-to-use Visual Studio report designer and a powerful API
- Offers seamless run-time deployment, royalty free

NEW RELEASE



## Janus WinForms Controls Suite V4.0 | from $853.44

Janus systems

**Add powerful Outlook style interfaces to your .NET applications.**

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



**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



JAMES MCCAFFREY

# Super-Simple Mutation Testing

Most testers I know have heard of mutation testing, but few have actually performed it. Mutation testing has the reputation of being difficult and requiring expensive third-party software tools. However, in this month's column, I'll show you how to create a super-simple (fewer than two pages of code and four hours of time) mutation testing system using C# and Visual Studio. By keeping the mutation testing system simple, you can get most of the benefits of a full-fledged mutation system at a fraction of the time and effort.

Mutation testing is a way to measure the effectiveness of a set of test cases. The idea is simple. Suppose you start with 100 test cases and your system under test (SUT) passes all 100 tests. If you mutate the SUT—for example, by changing a ">" to a "<" or by changing a "+" to a "-"—you presumably introduce a bug into the SUT. Now if you rerun your 100 test cases, you'd expect at least one test case failure indicating that one of your test cases caught the faulty code. But if you see no test failures, then it's quite likely that your set of test cases missed the faulty code and didn't thoroughly exercise the SUT.

The best way for you to see where I'm headed is by looking at **Figure 1**.

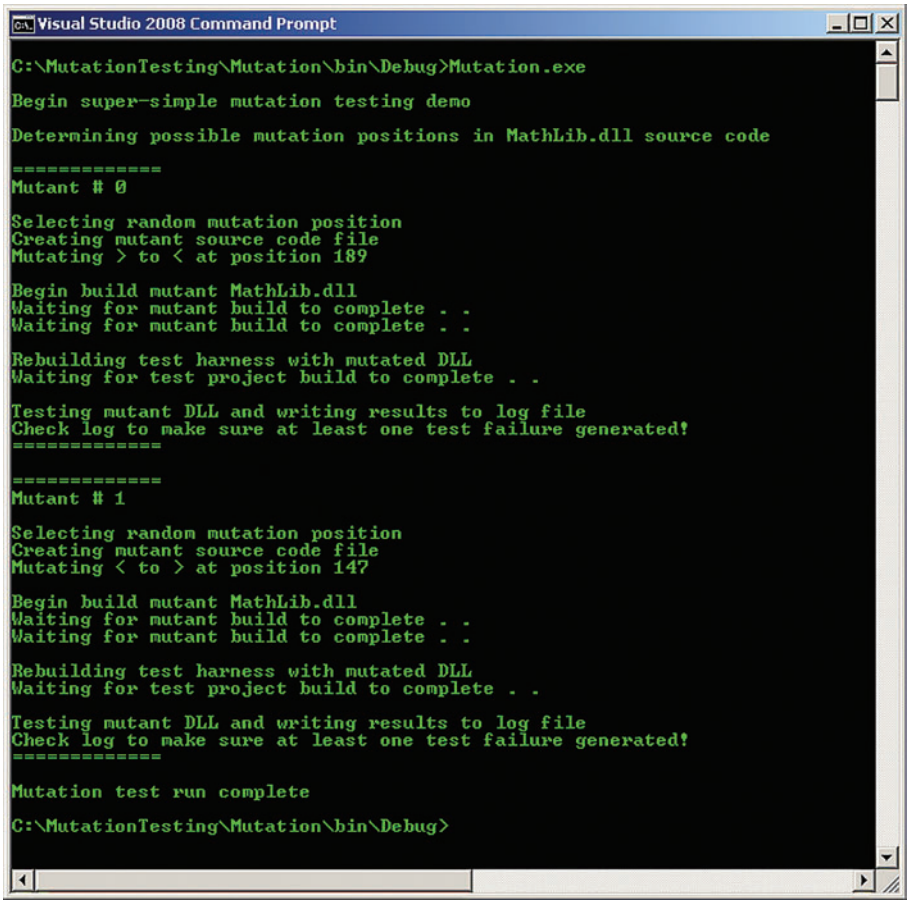



Figure 1 **Mutation Testing Demo Run**

The SUT in this example is a library named MathLib.dll. The technique I present here can be used to test most Microsoft .NET Framework systems including DLLs, WinForms applications, ASP.NET Web applications and so on. The mutation system begins by scanning the original source code for the SUT, looking for candidate code to mutate. My super-simple system looks only for "<" and ">" operators. The test system is set to create and evaluate two mutants. In a production scenario, you'd likely create hundreds or even thousands of mutants. The first mutant randomly selects an operator to mutate, in this case a ">" operator at character position 189 in the SUT source code, and mutates the token to "<". Next, the mutant DLL source code is built to create a mutant MathLb.dll library. Then the mutation system calls a suite of test cases on the mutant SUT, logging results to a file. The second iteration creates and tests a second mutant in the same way. The result of the log file is:

```
==========
Number failures = 0
Number test case failures = 0 indicates possible weak test suite!
==========
Number failures = 3
This is good.
==========
```

The first mutant didn't generate any test case failures, which means you should examine the source code at position 189 and determine why none of your test cases exercise that code.

Code download available at code.msdn.microsoft.com/mag201105TestRun.

## The SUT

My super-simple mutation testing demo consists of three Visual Studio projects. The first project holds the SUT, and in this case is a C# class library named MathLib. The second project is a test harness executable, in this case a C# console application named TestMutation. The third project creates and builds the mutants, in this case a C# console application named Mutation. For convenience I placed all three projects in a single directory named MutationTesting. With mutation testing there are a lot of files and folders to keep track of and you shouldn't underestimate the challenge of keeping them organized. For this demo I used Visual Studio 2008 (but any Visual Studio version will work) to create a dummy MathLib class library. The entire source code for the dummy SUT is shown in **Figure 2**.

Notice I retained the default class name of Class1. The class contains a single static method, TriMin, which returns the smallest of three type double parameters. Also note the SUT is deliberately incorrect. For example, if x = 2.0, y = 3.0 and z = 1.0, the TriMin method returns 2.0 instead of the correct 1.0 value. However, it's important to note that mutation testing does *not* directly measure the correctness of the SUT; mutation testing measures the effectiveness of a set of test cases. After building the SUT, the next step is to save a baseline copy of the source file, Class1.cs, to the root directory of the mutation testing system. The idea is that each mutant is a single modification of the original source code of the SUT and so a copy of the original SUT source must be maintained. In this example I saved the original source as Class1-Original.cs in directory C:\MutationTesting\Mutation.

## The Test Harness

In some testing situations, you may have an existing set of test-case data, and in some situations you have an existing test harness. For this super-simple mutation testing system, I created a C# console application test harness named TestMutation. After creating the project in Visual Studio, I added a Reference to the SUT: MathLib.dll located at C:\MutationTesting\MathLib\bin\Debug. The entire source code for the test harness project is presented in **Figure 3**.

> By keeping the mutation testing system simple, you can get most of the benefits of a full-fledged mutation system at a fraction of the time and effort.

Observe that the test harness has three hardcoded test cases. In a production environment, you'd likely have many hundreds of test cases stored in a text file and you could pass the filename in to Main as args[0]. The first test case, "1.0, 2.0, 3.0, 1.0," represents the x, y and z parameters (1.0, 2.0 and 3.0), followed by the expected result (1.0) for the TriMin method of the SUT. It's obvious the test set is inadequate: Each of the three test cases is basically equivalent and has the smallest value as the x parameter. But if you examine the original SUT, you'll see that all three test cases would in fact pass. Will our mutation testing system detect the weakness of the test set?

The test harness iterates through each test case, parses out the input parameters and the expected return value, calls the SUT with the input parameters, fetches the actual return value, compares the actual return with the expected return to determine a test case pass/fail result, and then accumulates the total number of test case failures. Recall that in mutation testing, we're primarily interested in whether there's at least one new failure, rather than how many test cases pass. The test harness writes the log file to the root folder of the calling program.

> Mutation testing is a way to measure the effectiveness of a set of test cases.

## The Mutation Testing System

In this section, I'll walk you through the mutation testing program one line at a time, but omit most of the WriteLine statements used to produce the output shown in **Figure 1**. I created a C# console application named Mutation in the root MutationTesting directory. The program begins with:

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Diagnostics;
using System.Threading;

namespace Mutation
{
  class Program
  {
    static Random ran = new Random(2);
    static void Main(string[] args)
    {
      try
      {
        Console.WriteLine("\nBegin super-simple mutation testing demo\n");
...
```

The purpose of the Random object is to generate a random mutation position. I used a seed value of 2, but any value will work fine. Next, I set up the file locations:

```
string originalSourceFile = "..\\..\\Class1-Original.cs";
string mutatedSourceFile = "..\\..\\..\\MathLib\\Class1.cs";
string mutantProject = "..\\..\\..\\MathLib\\MathLib.csproj";
string testProject = "..\\..\\..\\TestMutation\\TestMutation.csproj";
string testExecutable =
  "..\\..\\..\\TestMutation\\bin\\Debug\\TestMutation.exe";
string devenv =
  "C:\\Program Files (x86)\\Microsoft Visual Studio 9.0\\Common7\\IDE\\
  devenv.exe";
...
```

You'll see how each of these files is used shortly. Notice that I point to the devenv.exe program associated with Visual Studio 2008. Instead of hardcoding this location, I could have made a copy of devenv.exe and placed it inside the mutation system root folder.

The program continues:

```
List<int> positions = GetMutationPositions(originalSourceFile);
int numberMutants = 2;
...
```

I call a helper GetMutationPositions method to scan through the original source code file and store the character positions of all "<" and ">" characters into a List, and set the number of mutants to create and test to two.

The main processing loop is:

```
for (int i = 0; i < numberMutants; ++i) {
  Console.WriteLine("Mutant # " + i);
  int randomPosition = positions[ran.Next(0, positions.Count)];
  CreateMutantSource(originalSourceFile, randomPosition, mutatedSourceFile);

  try {
    BuildMutant(mutantProject, devenv);
    BuildTestProject(testProject, devenv);
    TestMutant(testExecutable);
  }
  catch {
    Console.WriteLine("Invalid mutant. Aborting.");
    continue;
  }
}
...
```

Inside the loop, the program fetches a random position of a character to mutate from the List of possible positions and then calls helper methods to generate mutant Class1.cs source code, build the corresponding mutant MathLib.dll, rebuild the test harness so that it uses the new mutant and then test the mutant DLL, hoping to generate an error. Because it's quite possible that mutated source code may not be valid, I wrap the attempt to build and test in a try-catch statement so I can abort the testing of non-buildable code.

The Main method wraps up as:

```
    ...
    Console.WriteLine("\nMutation test run complete");
  }
  catch (Exception ex) {
    Console.WriteLine("Fatal: " + ex.Message);
  }
} // Main()
```

## Creating Mutant Source Code

The helper method to get a list of possible mutation positions is:

```
static List<int> GetMutationPositions(string originalSourceFile)
{
  StreamReader sr = File.OpenText(originalSourceFile);
  int ch = 0; int pos = 0;
  List<int> list = new List<int>();
  while ((ch = sr.Read()) != -1) {
    if ((char)ch == '>' || (char)ch == '<')
      list.Add(pos);
    ++pos;
  }
  sr.Close();
  return list;
}
```

Figure 2 **The Entire Dummy SUT Source Code**

```
using System;
namespace MathLib
{
  public class Class1
  {
    public static double TriMin(double x, double y, double z)
    {
      if (x < y)
        return x;
      else if (z > y)
        return y;
      else
        return z;
    }
  }
}
```

Figure 3 **The Test Harness and Test Data**

```
using System;
using System.IO;

namespace TestMutation
{
  class Program
  {
    static void Main(string[] args)
    {
      string[] testCaseData = new string[]
        { "1.0, 2.0, 3.0, 1.0",
          "4.0, 5.0, 6.0, 4.0",
          "7.0, 8.0, 9.0, 7.0"};

      int numFail = 0;

      for (int i = 0; i < testCaseData.Length; ++i) {
        string[] tokens = testCaseData[i].Split(',');
        double x = double.Parse(tokens[0]);
        double y = double.Parse(tokens[1]);
        double z = double.Parse(tokens[2]);
        double expected = double.Parse(tokens[3]);

        double actual = MathLib.Class1.TriMin(x, y, z);
        if (actual != expected) ++numFail;
      }

      FileStream ofs = new FileStream("..\\..\\logFile.txt",
        FileMode.Append);
      StreamWriter sw = new StreamWriter(ofs);
      sw.WriteLine("=============");
      sw.WriteLine("Number failures = " + numFail);
      if (numFail == 0)
        sw.WriteLine(
          "Number test case failures = " +
          "0 indicates possible weak test suite!");
      else if (numFail > 0)
        sw.WriteLine("This is good.");
      sw.Close(); ofs.Close();
    }
  }
}
```

The method marches through the source code one character at a time looking for greater-than and less-than operators and adding the character position to a List collection. Notice that a limitation of this super-simple mutation system as presented is that it can only mutate single-character tokens such as ">" or "+" and can't deal with multicharacter tokens such as ">=". The helper method to actually mutate the SUT source code is listed in **Figure 4**.

> The technique I present here can be used to test most Microsoft .NET Framework systems.

The CreateMutantSource method accepts the original source code file, which was saved away earlier, along with a character position to mutate and the name and location of the resulting mutant file to save to. Here I just check for "<" and ">" characters, but you may want to consider other mutations. In general, you want mutations that will produce valid source, so, for example, you wouldn't change ">" to "=". Also, mutating in more than one location isn't a good idea, because just one of the mutations might generate a new test case failure, suggesting that the test set is good when in fact it might not be. Some mutations will have no practical effect (such as mutating a character

inside a comment), and some mutations will produce invalid code (such as changing the ">>" shift operator to "><").

## Building and Testing the Mutant

The BuildMutant helper method is:

```
static void BuildMutant(string mutantSolution, string devenv)
{
  ProcessStartInfo psi =
    new ProcessStartInfo(devenv, mutantSolution + " /rebuild");
  Process p = new Process();

  p.StartInfo = psi; p.Start();
  while (p.HasExited == false) {
    System.Threading.Thread.Sleep(400);
    Console.WriteLine("Waiting for mutant build to complete . . ");
  }
  p.Close();
}
```

I use a Process object to invoke the devenv.exe program to rebuild the Visual Studio solution that houses the Class1.cs mutated source code and produces the MathLib.dll mutant. Without arguments, devenv.exe launches the Visual Studio IDE, but when passed arguments, devenv can be used to rebuild Projects or Solutions. Notice I use a delay loop, pausing every 400 milliseconds, to give devenv.exe time to finish building the mutant DLL; otherwise the mutation system could attempt to test the mutant SUT before it's created.

The helper method to rebuild the test harness is:

```
static void BuildTestProject(string testProject, string devenv)
{
  ProcessStartInfo psi =
    new ProcessStartInfo(devenv, testProject + " /rebuild");
  Process p = new Process();

  p.StartInfo = psi; p.Start();
  while (p.HasExited == false) {
    System.Threading.Thread.Sleep(500);
    Console.WriteLine("Waiting for test project build to complete . . ");
  }
  p.Close();
}
```

Figure 4 **The CreateMutantSource Method**

```
static void CreateMutantSource(string originalSourceFile,
  int mutatePosition, string mutatedSourceFile)
{
  FileStream ifs = new FileStream(originalSourceFile, FileMode.Open);
  StreamReader sr = new StreamReader(ifs);
  FileStream ofs = new FileStream(mutatedSourceFile, FileMode.Create);
  StreamWriter sw = new StreamWriter(ofs);
  int currPos = 0;
  int currChar;

  while ((currChar = sr.Read()) != -1)
  {
    if (currPos == mutatePosition)
    {
      if ((char)currChar == '<') {
        sw.Write('>');
      }
      else if ((char)currChar == '>') {
        sw.Write('<');
      }
      else sw.Write((char)currChar);
    }
    else
       sw.Write((char)currChar);

    ++currPos;
   }

  sw.Close(); ofs.Close();
  sr.Close(); ifs.Close();
}
```

The main idea here is that, by rebuilding the test project, the new mutant SUT will be used when the test harness executes rather than the previously used mutant SUT. If your mutant source code is invalid, BuildTestProject will throw an Exception.

The last part of the super-simple mutation testing system is the helper method to invoke the test harness:

```
...
    static void TestMutant(string testExecutable)
    {
      ProcessStartInfo psi = new ProcessStartInfo(testExecutable);
      Process p = new Process(); p.StartInfo = psi;
      p.Start();
      while (p.HasExited == false)
        System.Threading.Thread.Sleep(200);

      p.Close();
    }

  } // class Program
} // ns Mutation
```

> Mutation testing is simple in principle, but the details of creating a full-fledged mutation testing system are challenging.

As I mentioned earlier, the test harness uses a hardcoded log file name and location; you could parameterize that by passing information as a parameter to TestMutant and placing it inside the Process start info, where it would be accepted by the TestMutation.exe test harness.

## A Real-World, Working Mutation Testing System

Mutation testing is simple in principle, but the details of creating a full-fledged mutation testing system are challenging. However, by keeping the mutation system as simple as possible and leveraging Visual Studio and devenv.exe, you can create a surprisingly effective mutation testing system for .NET SUTs. Using the example I've presented here, you should be able to create a mutation testing system for your own SUTs. The primary limitation of the sample mutation testing system is that, because the system is based on single-character changes, you can't easily perform mutations of multi-character operators, such as changing ">=" to its "<" complement operator. Another limitation is that the system only gives you the character position of the mutation, so it doesn't provide you with an easy way to diagnose a mutant. In spite of these limitations, my sample system has been used successfully to measure the effectiveness of test suites for several midsize software systems. ■

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

**THANKS** to the following Microsoft technical experts for reviewing this article: *Paul Koch, Dan Liebling and Shane Williams*

# 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

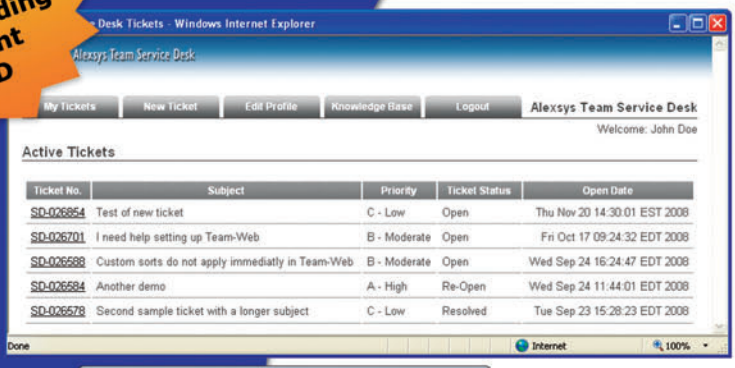
### Service Desk Features

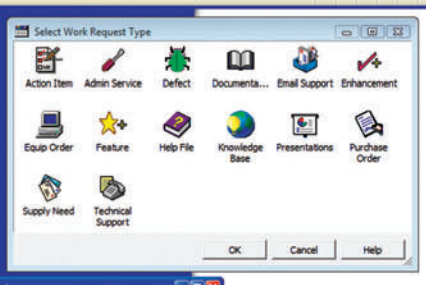
- Fully Secure
- Unlimited Users Self Registered or Active Directory
- Integrated into Your Web Site
- Fast/AJAX Dynamic Content
- Unlimited Service Desks
- Visual Service Desk Builder

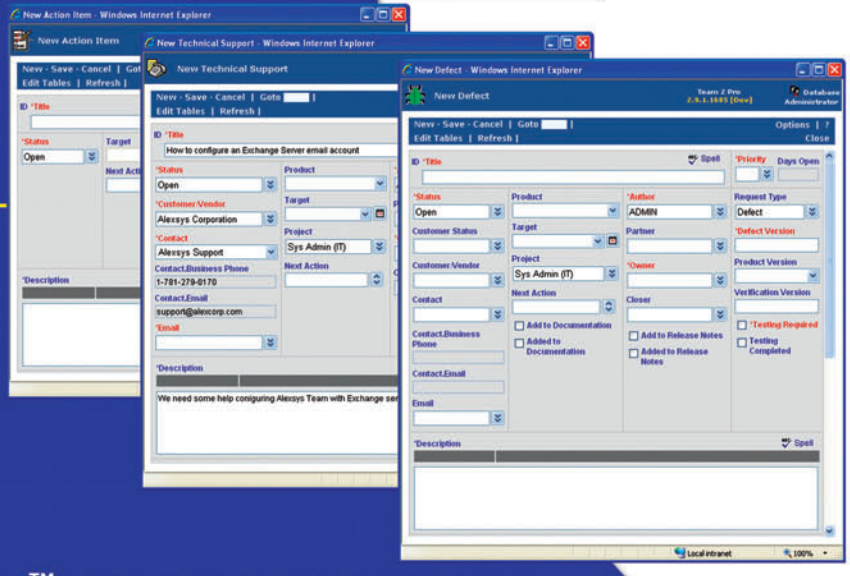
### Team 2 Features

- 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.



TED NEWARD

# Multiparadigmatic .NET, Part 7: Parametric Metaprogramming

When I was a university student, in a calculus-filled microeconomics lecture, the professor shared some words of wisdom that resonate with me to this day:

"If, while slogging through the tedious details of our chosen subject matter, you find yourself unable to see the reason behind why we're slogging through all these tedious details, your responsibility is to interrupt me and say, 'Professor Anderson, what's the point?' And we'll take a few moments to go back and explain how we got here."

Readers who've been sitting through all the articles in this series may well have hit that wall, so let's take a few moments to go back and review how we got here.

## Recap

At its heart, as described by James Coplien in his book "Multi-Paradigm Design for C++" (Addison-Wesley, 1998), which inspired much of the writing in this series of articles, all programming is an exercise in capturing commonality—writing code that represents the "all the time" case—and then using the variability constructs within the language to allow it to behave or be structured differently under certain circumstances.

Object-oriented programming, for example, captures commonality into classes, then permits variability through inheritance, the creation of subclasses that alter that commonality. Typically that's done by changing the behavior of certain parts of the class using methods or messages, depending on the language in question. The commonality and variability needs of the project don't always fit into the object-oriented paradigm so neatly, however, or any other particular single paradigm, for that matter—object-oriented programming grew out of procedural programming as an attempt to offer variability that procedural programming couldn't capture easily.

Fortunately for readers of this magazine, the languages offered by Microsoft in Visual Studio 2010 are multiparadigmatic languages, meaning that they draw several different programming paradigms together into a single language. Coplien first identified C++ as such a multiparadigmatic language, in that it brought three major paradigms together: procedural, object and metaprogrammatic (sometimes also more accurately referred to as metaobject). C++ was also widely criticized as a complicated language, too difficult for the average developer to master, mostly because it was hard to see when to use the various features of the language to solve particular problems.

Modern languages frequently develop into highly multiparadigmatic languages. F#, C# and Visual Basic, as of Visual Studio 2010, support five such paradigms directly: procedural, object-oriented, metaobject, dynamic and functional. All three languages—four, if you include C++/CLI in that mix—therefore run the risk of falling into the same fate as C++.

Without a clear understanding of each of the paradigms mixed into those languages, developers can easily run afoul of the favorite feature trap, where developers rely too much on one feature or paradigm to the exclusion of the others and end up creating overly complex code that eventually gets tossed and rewritten. Do this too many times and the language starts to bear the brunt of developer frustration, leading eventually to calls for a new language, or just retirement in general.

## Procedural and Object-Oriented Paradigms

Thus far, we've seen the commonality/variability analysis applied to procedural or structural programming, wherein we capture commonality in data structures and operate on those structures by feeding them into different procedure calls, creating variability by creating new procedure calls to operate on those same data structures. We've also seen commonality/variability around objects, in which we capture commonality into classes and then create variability by subclassing those classes and changing bits and pieces of them through overriding methods or properties.

The languages offered by Microsoft in Visual Studio 2010 are multiparadigmatic languages.

Bear in mind, too, that another problem emerges in that inheritance (for the most part) only allows for positive variability—we can't remove something from a base class, such as member method or field. In the CLR, we can hide derived-accessible member implementation by shadowing it (using the virtual keyword instead of the override keyword in C#, for example). However, that means replacing its behavior with something else, not removing it outright. Fields remain present regardless.

This observation leads to a disturbing revelation for some: Objects can't do everything we need—at least not pure objects. For example, objects can't capture variability along structural lines using inheritance: a collection class that captures stack-like behavior, but for a

variety of different data types (integers, doubles, strings and so on) can't capture that structural difference. Granted, within the CLR we can use the unified type system. We can store reference instances of System.Object and downcast as necessary, but that's not the same as being able to create a type that stores only one type.

This realization brought us to the subject of metaobject programming, as we're looking for ways to capture things outside of traditional object axes.

> Generics permit the compile-time creation of types that have parts of their structure supplied at compile time from client code.

The first such meta approach was generative, wherein source code was generated based on some kind of template. This approach allows for some variability on a variety of different axes, but is limited (for the most part) to a source-time execution model. It also begins to break down as the number of variabilities rises because the source template has to somehow vary the code generated (typically with decision-making statements hiding inside the template language) at code-generation time, and this can create complexities in the templates.

The second such meta approach was that of reflective or attributive programming. At run time, the code uses the full-fidelity metadata facilities of the platform (Reflection) to inspect code and behave differently based on what it sees there.

This permitted flexibility of implementation or behavior around runtime decision-making, but introduced its own limitations: no type relationships are present within a reflective/attributive design, meaning there's no way to programmatically ensure that only database-persistable types can be passed into a method, for example, as opposed to XML-persistable types. The lack of an inheritance or other relationship means that a certain amount of type-safety (and therefore an important ability to prevent errors) is thus lost.

Which brings us to the third metaobject facility within the .NET environment: parametric polymorphism. This means the ability to define types that have types as parameters. Or, put more simply, what the Microsoft .NET Framework refers to as generics.

## Generics

In its simplest form, generics permit the compile-time creation of types that have parts of their structure supplied at compile time from client code. In other words, the developer of a stack behavioral collection doesn't have to know at the time his library is compiled what kinds of types his clients might want to store in different instances—they supply that information when they create instances of the collection.

For example, in a previous article, we saw that the definition of a Cartesian point type requires an ahead-of-time decision (on the part of the Point developer) on the representation of the axis values (X and Y). Should they be integral values? Should they be allowed to be negative?

A Cartesian point used in mathematics could very well need to be a floating point and negative. A Cartesian point used to represent a pixel on a computer graphics screen needs to be positive, integral and likely within a certain numerical range, because computer displays of 4 billion by 4 billion are not yet commonplace.

Thus, on the surface of it, a well-design Cartesian point library will need several different Point types: one using unsigned bytes as X and Y fields, one using doubles as X and Y fields and so on. The behavior, for the most part, will be identical across all of these types, clearly highlighting a violation of the desire to capture commonality (known colloquially as the DRY Principle: "Don't Repeat Yourself").

Using parametric polymorphism, we can capture that commonality quite neatly:

```
class Point2D<T> {
  public Point2D(T x, T y) { this.X = x; this.Y = y; }

  public T X { get; private set; }
  public T Y { get; private set; }
  // Other methods left to the reader's imagination
}
```

Now the developer can specify precisely the range and type properties of the Cartesian point he wishes to use. When working in a mathematical domain, he creates instances of Point2D<double> values, and when working to display those values to the screen, he creates instances of Point2D<sbyte> or Point2D<ushort>. Each is its own distinct type, so attempts to compare or assign Point2D<sbyte> to Point2D<double> will fail miserably at compile time, exactly as a strongly typed language would prefer.

As written, however, the Point2D type still has some drawbacks. We've captured the commonality of Cartesian points, certainly, but we've essentially allowed for any kind of values to be used for the X and Y values. While this could conceivably be useful in certain scenarios ("In this graph, we're charting the ratings each person gave a particular movie"), as a general rule, trying to create a Point2D<DateTime> is potentially confusing, and trying to create a Point2D<System.Windows.Forms.Form> almost certainly is. We need to introduce some kind of negative variability here (or, if you prefer, throttle back the degree of positive variability), restricting the kinds of types that can be axis values in a Point2D.

Many .NET languages capture this negative variability via parameterization constraints—also sometimes referred to as type constraints—by explicitly describing conditions the type parameter must have:

Figure 1 **Conversion Is in Order**

```
class USD { }
class EUR { }
class Money<C> {
  public float Quantity { get; set; }
  public C Currency { get; set; }

  public static Money<C> operator +(
    Money<C> lhs, Money<C> rhs) {
    return new Money<C>() {
      Quantity = lhs.Quantity + rhs.Quantity,
      Currency = lhs.Currency };
  }
}
...
var pizza = new Money<USD>() {
  Quantity = 4.99f, Currency = new USD() };
var beer = new Money<EUR>() {
  Quantity = 3.5f, Currency = new EUR() };
var lunch = pizza + beer;    // ERROR
```

```
class Point2D<T> where T : struct {
  public Point2D(T x, T y) { this.X = x; this.Y = y; }

  public T X { get; private set; }
  public T Y { get; private set; }
  // Other methods left to the reader's imagination
}
```

This means the compiler won't accept anything for T that isn't a value type.

To be honest, this isn't exactly a negative variability, per se, but it serves as one compared to the problem of trying to remove certain functionality, which approximates a fair amount of what a true negative variability would.

## Varying Behavior

Parametric polymorphism is typically used to provide variability on the structural axis, but as the developers of the C++ Boost libraries demonstrated, it's not the only axis along which it can operate. With judicious use of type constraints, we can also use generics to provide a policy mechanism in which clients can specify a behavioral mechanism for objects being constructed.

Consider, for a moment, the traditional problem of diagnostic logging: To help diagnose problems with code running on a server (or even on client machines), we want to track the execution of code through the codebase. This typically means writing messages to file. But sometimes we want the messages to appear on the console, at least for certain scenarios, and sometimes we want the messages thrown away. Handling diagnostic logging messages has been a tricky problem over the years, and a variety of solutions have been proposed. The lessons of Boost offer a new approach as well.

We start by defining an interface:

```
interface ILoggerPolicy {
  void Log(string msg);
}
```

It's a straightforward interface, with one or more methods defining the behavior we want to vary, which we do via a series of subtypes of that interface:

```
class ConsoleLogger : ILoggerPolicy {
  public void Log(string msg) { Console.WriteLine(msg); }
}

class NullLogger : ILoggerPolicy {
  public void Log(string msg) { }
}
```

Figure 2 **Combining Types Intentionally**

```
class USD { }
class EUR { }
class Money<C> {
  public float Quantity { get; set; }
  public C Currency { get; set; }

  public static Money<C> operator +(
    Money<C> lhs, Money<C> rhs) {
    return new Money<C>() {
      Quantity = lhs.Quantity + rhs.Quantity,
      Currency = lhs.Currency };
  }

  public Money<C2> Convert<C2>() where C2 : new() {
    return new Money<C2>() { Quantity =
      this.Quantity, Currency = new C2() };
  }
}
```

Here we have two possible implementations, one of which writes the log message to the console while the other throws it away.

Using this requires clients to opt in by declaring the logger as a typed parameter, and creating an instance of it to do the actual logging:

```
class Person<A> where A : ILoggerPolicy, new() {
  public Person(string fn, string ln, int a) {
    this.FirstName = fn; this.LastName = ln; this.Age = a;
    logger.Log("Constructing Person instance");
  }

  public string FirstName { get; private set; }
  public string LastName { get; private set; }
  public int Age { get; private set; }

  private A logger = new A();
}
```

Describing which logger type to use, then, is a simple matter of passing a constructor-time parameter, like so:

```
Person<ConsoleLogger> ted =
  new Person<ConsoleLogger>("Ted", "Neward", 40);
var anotherTed  =
  new Person<NullLogger>("Ted", "Neward", 40);
```

This mechanism allows developers to create their own custom logging implementations, and plug them in to be used by Person<> instances without the Person<> developer having to know any details of the logging implementation used. But numerous other approaches also do this, such as having a Logger field or property that passes in a Logger instance from outside (or obtaining one through a Dependency-Injection approach). The generics-based approach has one advantage that the field-based approach doesn't, however, and that's compile-time distinction: a Person<ConsoleLogger> is a distinct and separate type from a Person<NullLogger>.

> Quantities are useless without the units being quantified.

## Money, Money, Money

One problem that plagues developers is that quantities are useless without the units being quantified. One thousand pennies is clearly not the same thing as 1,000 horses or 1,000 employees, or 1,000 pizzas. Yet, just as clearly, 1,000 pennies and 10 dollars are, in fact, the same value.

This becomes even more important in mathematical calculations where the need to capture the units (degrees/radians, feet/meters, Fahrenheit/Celsius) is even more critical, particularly if you're writing guidance control software for a really big rocket. Consider the Ariane 5, whose maiden flight had to be self-destructed due to an error in conversion. Or the NASA probe to Mars, one of which slammed into the Martian landscape at full speed due to a conversion error.

Recently, new languages like F# have decided to acclimate units of measure as a direct language feature, but even C# and Visual Basic can do similar kinds of things, thanks to generics.

Channeling our inner Martin Fowler, let's start with a simple Money class, which knows the amount (quantity) and currency (type) of a particular monetary amount:

```
class Money {
  public float Quantity { get; set; }
  public string Currency { get; set; }
}
```

On the surface, this seems workable, but before too long we're going to want to start doing value-like things with this, such as add Money instances together (a pretty common thing to do with money, when you think about it):

```
class Money {
  public float Quantity { get; set; }
  public string Currency { get; set; }

  public static Money operator +(Money lhs, Money rhs) {
    return new Money() {
      Quantity = lhs.Quantity + rhs.Quantity, Currency = lhs.Currency };
  }
}
```

Of course, the problem is going to arise when we try to add U.S. dollars (USD) and European euro (EUR) together, such as when we go out to lunch (after all, everybody knows Europeans brew the best beer, but Americans make the best pizza):

```
var pizza = new Money() {
  Quantity = 4.99f, Currency = "USD" };
var beer = new Money() {
  Quantity = 3.5f, Currency = "EUR" };
var lunch = pizza + beer;
```

Anybody who takes a quick glance at the financial dashboards is going to realize that somebody's getting ripped off—the euros are being converted to dollars at a 1-1 ratio. In order to prevent accidental fraud, we probably want to make sure the compiler knows not to convert USD to EUR without going through an approved conversion process that looks up the current conversion rate (see **Figure 1**).

Notice how USD and EUR are basically just placeholders, designed to give the compiler something to compare against. If the two C type parameters aren't the same, it's a problem.

Of course, we've also lost the ability to combine the two, and there will be times when we want to do exactly that. Doing so requires a bit more parametric syntax (see **Figure 2**).

This is a specialized generic method within a generic class, and the <> syntax after the method name adds more type parameters to the scope of the method—in this case, the type of the second currency to convert over to. So, buying a pizza and beer now becomes something like:

```
var pizza = new Money<USD>() {
  Quantity = 4.99f, Currency = new USD() };
var beer = new Money<EUR>() {
  Quantity = 3.5f, Currency = new EUR() };
var lunch = pizza + beer.Convert<USD>();
```

If desirable, we could even use the conversion operator (in C#) to do the conversion automatically, but that could potentially be more confusing than helpful to readers of the code, depending on your aesthetic preferences.

## Wrapping Up

What's missing from the Money<> example is obvious: clearly there needs to be some way to convert dollars to euros and euros to dollars. But part of the goal in designs like this is to avoid a closed system—that is, as new currencies are needed (rubles, rupees, pounds, lira or whatever else floats your monetary boat), it would be nice if we, the original designers of the Money<> type, don't have to be called to add them. Ideally, in an open system, other developers can plug them in as they need and everything "just works."

> If the two C type parameters aren't the same, it's a problem.

Don't tune out just yet, though, and certainly don't start shipping the code as is. We still have a few adjustments to make to the Money<> type to make it more powerful, safe and extensible. Along the way, we'll have a look at dynamic and functional programming.

But for now, happy coding! ■

**TED NEWARD** *is a principal with Neward & Associates, an independent firm specializing in enterprise .NET Framework and Java platform systems. He has written more than 100 articles, is a C# MVP and INETA speaker, and has authored or coauthored a dozen books, including "Professional F# 2.0" (Wrox, 2010). He consults and mentors regularly—reach him at ted@tedneward.com if you're interested in having him come work with your team, or read his blog at blogs.tedneward.com.*

**THANKS** to the following technical expert for reviewing this article: *Krzysztof Cwalina*



CHARLES PETZOLD

# Silverlight Printing Basics

Silverlight 4 added printing to the Silverlight feature list, and I want to plunge right in by showing you a tiny program that put a big smile on my face.

The program is called PrintEllipse and that's all it does. The XAML file for MainPage contains a Button, and **Figure 1** shows the MainPage codebehind file in its entirety.

Notice the using directive for System.Windows.Printing. When you click the button, the program creates an object of type PrintDocument and assigns a handler for the PrintPage event. When the program calls the Print method, the standard print dialog box appears. The user can take this opportunity to set which printer to use and set various properties for printing, such as portrait or landscape mode.

When the user clicks Print on the print dialog, the program receives a call to the PrintPage event handler. This particular program responds by creating an Ellipse element and setting that to the PageVisual property of the event arguments. (I deliberately chose light pastel colors so the program won't use too much of your ink.) Soon a page will emerge from your printer filled with a giant ellipse.

You can run this program from my Web site at bit.ly/dU9B7k and check it out yourself. All the source code from this article is also downloadable, of course.

If your printer is like most printers, the internal hardware prohibits it from printing to the very edge of the paper. Printers usually have an intrinsic built-in margin in which nothing is printed; printing is instead restricted to a "printable area" that's less than the full size of the page.

What you'll notice with this program is that the ellipse appears in its entirety within the printable area of the page, and obviously this happens with minimum effort on the part of the program. The printable area of the page behaves much like a container element on the screen: It only clips a child when an element has a size that exceeds the area. Some far more sophisticated graphics environments—such as Windows Presentation Foundation (WPF)—don't behave nearly as well (but, of course, WPF offers much more printing control and flexibility than Silverlight).

Figure 1 **The MainPage Code for PrintEllipse**

```
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Media;
using System.Windows.Printing;
using System.Windows.Shapes;

namespace PrintEllipse
{
  public partial class MainPage : UserControl
  {
    public MainPage()
    {
      InitializeComponent();
    }
    void OnButtonClick(object sender, RoutedEventArgs args)
    {
      PrintDocument printDoc = new PrintDocument();
      printDoc.PrintPage += OnPrintPage;
      printDoc.Print("Print Ellipse");
    }
    void OnPrintPage(object sender, PrintPageEventArgs args)
    {
      Ellipse ellipse = new Ellipse
      {
        Fill = new SolidColorBrush(Color.FromArgb(255, 255, 192, 192)),
        Stroke = new SolidColorBrush(Color.FromArgb(255, 192, 192, 255)),
        StrokeThickness = 24    // 1/4 inch
      };
      args.PageVisual = ellipse;
    }
  }
}
```

## PrintDocument and Events

Besides the PrintPage event, PrintDocument also defines BeginPrint and EndPrint events, but these aren't nearly as important as PrintPage. The BeginPrint event signals the beginning of a print job. It's fired when the user exits the standard print dialog by pressing the Print button and gives the program the opportunity to perform initialization. The call to the BeginPrint handler is then followed by the first call to the PrintPage handler.

A program that wishes to print more than one page in a particular print job can do so. In every call to the PrintPage handler, the HasMorePages property of PrintPageEventArgs is initially set to false. When the handler is finished with a page, it can simply set the property to true to signal that at least one more page must be printed. PrintPage is then called again. The PrintDocument object maintains a PrintedPageCount property that's incremented following every call to the PrintPage handler.

When the PrintPage handler exits with HasMorePages set to its default value of false, the print job is over and the EndPrint event is fired, giving the program the opportunity to perform cleanup chores. The EndPrint event is also fired when an error occurs during the printing process; the Error property of EndPrintEventArgs is of type Exception.

## Printer Coordinates

The code shown in **Figure 1** sets the StrokeThickness of the Ellipse to 24, and if you measure the printed result, you'll discover that

Code download available at code.msdn.microsoft.com/mag201105UIFrontiers.

Figure 2 **The OnPrintPage Method to Calculate Margins**

```
void OnPrintPage(object sender, PrintPageEventArgs args)
{
  Thickness margin = new Thickness
  {
    Left = Math.Max(0, 96 - args.PageMargins.Left),
    Top = Math.Max(0, 96 - args.PageMargins.Top),
    Right = Math.Max(0, 96 - args.PageMargins.Right),
    Bottom = Math.Max(0, 96 - args.PageMargins.Bottom)
  };
  Ellipse ellipse = new Ellipse
  {
    Fill = new SolidColorBrush(Color.FromArgb(255, 255, 192, 192)),
    Stroke = new SolidColorBrush(Color.FromArgb(255, 192, 192, 255)),
    StrokeThickness = 24,   // 1/4 inch
    Margin = margin
  };
  Border border = new Border();
  border.Child = ellipse;
  args.PageVisual = border;
}
```

it's one-quarter inch wide. As you know, a Silverlight program normally sizes graphical objects and controls entirely in units of pixels. However, when the printer is involved, coordinates and sizes are in device-independent units of 1/96th inch. Regardless of the actual resolution of the printer, from a Silverlight program the printer always appears to be a 96 DPI device.

As you might know, this coordinate system of 96 units to the inch is used throughout WPF, where the units are sometimes referred to as "device-independent pixels." This value of 96 DPI wasn't chosen arbitrarily: By default, Windows assumes that your video display has 96 dots to the inch, so in many cases a WPF program is actually drawing in units of pixels. The CSS specification assumes that video displays have a 96 DPI resolution, and that value is used for converting between pixels, inches and millimeters. The value of 96 is also a convenient number for converting font sizes, which are commonly specified in points, or 1/72nd inch. A point is three-quarters of a device-independent pixel.

PrintPageEventArgs has two handy get-only properties that also report sizes in units of 1/96th inch: PrintableArea of type Size provides the dimensions of the area of the printable area of the page, and PageMargins of type Thickness is the width of the left, top, right and bottom of the unprintable edges. Add these two together (in the right way) and you get the full size of the paper.

My printer—when loaded with standard 8.5 x 11 inch paper and set for portrait mode—reports a PrintableArea of 791 x 993. The four values of the PageMargins property are 12 (left), 6 (top), 12 (right) and 56 (bottom). If you sum the horizontal values of 791, 12 and 12, you'll get 815. The vertical values are 994, 6 and 56, which sum to 1,055. I'm not sure why there's a one-unit difference between these values and the values of 816 and 1,056 obtained by multiplying the page size in inches by 96.

When a printer is set for landscape mode, then the horizontal and vertical dimensions reported by PrintableArea and PageMargins are swapped. Indeed, examining the PrintableArea property is the only way a Silverlight program can determine whether the printer is in portrait or landscape mode. Anything printed by the program is automatically aligned and rotated depending on this mode.

Often when you print something in real life, you'll define margins that are somewhat larger than the unprintable margins. How do you do this in Silverlight? At first, I thought it would be as easy as setting the Margin property on the element you're

Figure 3 **Printing an Image in PrintImage**

```
void OnPrintPage(object sender, PrintPageEventArgs args)
{
  // Find the full size of the page
  Size pageSize =
    new Size(args.PrintableArea.Width
    + args.PageMargins.Left + args.PageMargins.Right,
    args.PrintableArea.Height
    + args.PageMargins.Top + args.PageMargins.Bottom);

  // Get additional margins to bring the total to MARGIN (= 96)
  Thickness additionalMargin = new Thickness
  {
    Left = Math.Max(0, MARGIN - args.PageMargins.Left),
    Top = Math.Max(0, MARGIN - args.PageMargins.Top),
    Right = Math.Max(0, MARGIN - args.PageMargins.Right),
    Bottom = Math.Max(0, MARGIN - args.PageMargins.Bottom)
  };

  // Find the area for display purposes
  Size displayArea =
    new Size(args.PrintableArea.Width
    - additionalMargin.Left - additionalMargin.Right,
    args.PrintableArea.Height
    - additionalMargin.Top - additionalMargin.Bottom);

  bool pageIsLandscape = displayArea.Width > displayArea.Height;
  bool imageIsLandscape = bitmap.PixelWidth > bitmap.PixelHeight;

  double displayAspectRatio = displayArea.Width / displayArea.Height;
  double imageAspectRatio = (double)bitmap.PixelWidth / bitmap.PixelHeight;

  double scaleX = Math.Min(1, imageAspectRatio / displayAspectRatio);
  double scaleY = Math.Min(1, displayAspectRatio / imageAspectRatio);

  // Calculate the transform matrix
  MatrixTransform transform = new MatrixTransform();

  if (pageIsLandscape == imageIsLandscape)
  {
    // Pure scaling
    transform.Matrix = new Matrix(scaleX, 0, 0, scaleY, 0, 0);
  }
  else
  {
    // Scaling with rotation
    scaleX *= pageIsLandscape ? displayAspectRatio : 1 /
      displayAspectRatio;
    scaleY *= pageIsLandscape ? displayAspectRatio : 1 /
      displayAspectRatio;
    transform.Matrix = new Matrix(0, scaleX, -scaleY, 0, 0, 0);
  }

  Image image = new Image
  {
    Source = bitmap,
    Stretch = Stretch.Fill,
    Width = displayArea.Width,
    Height = displayArea.Height,
    RenderTransform = transform,
    RenderTransformOrigin = new Point(0.5, 0.5),
    HorizontalAlignment = HorizontalAlignment.Center,
    VerticalAlignment = VerticalAlignment.Center,
    Margin = additionalMargin,
  };

  Border border = new Border
  {
    Child = image,
  };

  args.PageVisual = border;
}
```

printing. This Margin would be calculated by starting with a desired total margin (in units of 1/96th inch) and subtracting the values of the PageMargins property available from the PrintPageEventArgs. That approach didn't work well, but the correct solution was almost as easy. The PrintEllipseWithMargins program (which you can run at bit.ly/fCBs3X) is the same as the first program except that a Margin property is set on the Ellipse, and then the Ellipse is set as the child of a Border, which fills the printable area. Alternatively, you can set the Padding property on the Border. **Figure 2** shows the new OnPrintPage method.

Figure 4 **The PrintCalendar Button**

## The PageVisual Object

There are no special graphics methods or graphics classes associated with the printer. You "draw" something on the printer page the same way you "draw" something on the video display, which is by assembling a visual tree of objects that derive from FrameworkElement. This tree can include Panel elements, including Canvas. To print that visual tree, set the topmost element to the PageVisual property of the PrintPageEventArgs. (PageVisual is defined as a UIElement, which is the parent class to FrameworkElement, but in a practical sense, everything you'll be setting to PageVisual will derive from FrameworkElement.)

Almost every class that derives from FrameworkElement has non-trivial implementations of the MeasureOverride and ArrangeOverride methods for layout purposes. In its MeasureOverride method, an element determines its desired size, sometimes by determining the desired sizes of its children by calling its children's Measure methods. In the ArrangeOverride method, an element arranges its children relative to itself by calling the children's Arrange methods.

When you set an element to the PageVisual property of PrintPageEventArgs, the Silverlight printing system calls Measure on that topmost element with the PrintableArea size. This is how (for example) the Ellipse or Border is automatically sized to the printable area of the page.

However, you can also set that PageVisual property to an element that's already part of a visual tree being displayed in the program's window. In this case, the printing system doesn't call Measure on that element, but instead uses the measurements and layout already determined for the video display. This allows you to print something from your program's window with reasonable fidelity, but it also means that what you print might be cropped to the size of the page.

You can, of course, set explicit Width and Height properties on the elements you print, and you can use the PrintableArea size to help out.

## Scaling and Rotating

The next program I took on turned out to be more of a challenge than I anticipated. The goal was a program that would let the user print any image file supported by Silverlight—namely PNG and JPEG files—stored on the user's local machine. This program uses the OpenFileDialog class to load these files. For security purposes, OpenFileDialog only returns a FileInfo object that lets the program open the file. No filename or directory is provided.

I wanted this program to print the bitmap as large as possible on the page (excluding a preset margin) without altering the bitmap's aspect ratio. Normally this is a snap: The Image element's default Stretch mode is Uniform, which means the bitmap is stretched as large as possible without distortion.

However, I decided that I didn't want to require the user to specifically set portrait or landscape mode on the printer commensurate with the particular image. If the printer was set to portrait mode, and the image was wider than its height, I wanted the image to be printed sideways on the portrait page. This little feature immediately made the program much more complex.

If I were writing a WPF program to do this, the program itself could have switched the printer into portrait or landscape mode. But that isn't possible in Silverlight. The printer interface is defined so that only the user can change settings like that.

Again, if I were writing a WPF program, alternatively I could have set a LayoutTransform on the Image element to rotate it 90 degrees. The rotated Image element would then be resized to fit on the page, and the bitmap itself would have been adjusted to fit the Image element.

But Silverlight doesn't support LayoutTransform. Silverlight only supports RenderTransform, so if the Image element must be rotated to accommodate a landscape image printed in portrait mode, the Image element must also be manually sized to the dimensions of the landscape page.

You can try out my first attempt at bit.ly/eMHOsB. The OnPrintPage method creates an Image element and sets the Stretch property to None, which means the Image element displays the bitmap in its pixel size, which on the printer means that each pixel is assumed to be 1/96th inch. The program then rotates, sizes and translates

Figure 5 **The CalendarPage Layout**

```
<UserControl x:Class="PrintCalendar.CalendarPage"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  FontSize="36">
  <Grid x:Name="LayoutRoot" Background="White">
    <Grid.RowDefinitions>
      <RowDefinition Height="Auto" />
      <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    <TextBlock Name="monthYearText"
      Grid.Row="0"
       FontSize="48"
       HorizontalAlignment="Center" />
    <Grid Name="dayGrid"
      Grid.Row="1">
      <Grid.ColumnDefinitions>
        <ColumnDefinition Width="*" />
        <ColumnDefinition Width="*" />
        <ColumnDefinition Width="*" />
        <ColumnDefinition Width="*" />
        <ColumnDefinition Width="*" />
        <ColumnDefinition Width="*" />
        <ColumnDefinition Width="*" />
      </Grid.ColumnDefinitions>
      <Grid.RowDefinitions>
        <RowDefinition Height="*" />
        <RowDefinition Height="*" />
        <RowDefinition Height="*" />
        <RowDefinition Height="*" />
        <RowDefinition Height="*" />
        <RowDefinition Height="*" />
      </Grid.RowDefinitions>
    </Grid>
  </Grid>
</UserControl>
```

Figure 6 **The CalendarPage Codebehind Constructor**

```
public CalendarPage(DateTime date)
{
  InitializeComponent();
  monthYearText.Text = date.ToString("MMMM yyyy");
  int row = 0;
  int col = (int)new DateTime(date.Year, date.Month, 1).DayOfWeek;
  for (int day = 0; day < DateTime.DaysInMonth(date.Year, date.Month); day++)
  {
    TextBlock txtblk = new TextBlock
    {
      Text = (day + 1).ToString(),
      HorizontalAlignment = HorizontalAlignment.Left,
      VerticalAlignment = VerticalAlignment.Top
    };
    Border border = new Border
    {
      BorderBrush = blackBrush,
      BorderThickness = new Thickness(2),
      Child = txtblk
    };
    Grid.SetRow(border, row);
    Grid.SetColumn(border, col);
    dayGrid.Children.Add(border);
    if (++col == 7)
    {
      col = 0;
      row++;
    }
  }
  if (col == 0)
    row--;
  if (row < 5)
    dayGrid.RowDefinitions.RemoveAt(0);
  if (row < 4)
    dayGrid.RowDefinitions.RemoveAt(0);
}
```

that Image element by calculating a transform that it applies to the RenderTransform property of the Image element.

The hard part of such code is, of course, the math, so it was pleasant to see the program work with portrait and landscape images with the printer set to portrait and landscape modes.

However, it was particularly unpleasant to see the program fail for large images. You can try it yourself with images that have dimensions somewhat greater (when divided by 96) than the size of the page in inches. The image is displayed at the correct size, but not in its entirety.

What's going on here? Well, it's something I've seen before on video displays. Keep in mind that the RenderTransform affects only how the element is displayed and not how it appears to the layout system. To the layout system, I'm displaying a bitmap in an Image element with Stretch set to None, meaning that the Image element is as large as the bitmap itself. If the bitmap is larger than the printer page, then some of that Image element need not be rendered, and it will, in fact, be clipped, regardless of a RenderTransform that's shrinking the Image element appropriately.

My second attempt, which you can try out at bit.ly/g4HJ1C, takes a somewhat different strategy. The OnPrintPage method is shown in **Figure 3**. The Image element is given explicit Width and Height settings that make it exactly the size of the calculated display area. Because it's all within the printable area of the page, nothing will be clipped. The Stretch mode is set to Fill, which means that the bitmap fills the Image element regardless of the aspect ratio. If the Image element won't be rotated, one dimension is correctly sized, and the other dimension must have a scaling factor applied that reduces the size. If the Image element must also be rotated, then the scaling factors must accommodate the different aspect ratio of the rotated Image element.

The code is certainly messy—and I suspect there might be simplifications not immediately obvious to me—but it works for bitmaps of all sizes.

Another approach is to rotate the bitmap itself rather than the Image element. Create a WriteableBitmap from the loaded BitmapImage object, and a second WritableBitmap with swapped horizontal and vertical dimensions. Then copy all the pixels from the first WriteableBitmap into the second with rows and columns swapped.

## Multiple Calendar Pages

Deriving from UserControl is an extremely popular technique in Silverlight programming to create a reusable control without a lot of hassle. Much of a UserControl is a visual tree defined in XAML.

You can also derive from UserControl to define a visual tree for printing! This technique is illustrated in the PrintCalendar program, which you can try out at bit.ly/dlwSsn. You enter a start month and an end month, and the program prints all the months in that range, one month to a page. You can tape the pages to your walls and mark them up, just like a real wall calendar.

After my experience with the PrintImage program, I didn't want to bother with margins or orientation; instead, I included a Button that places the responsibility on the user, as shown in **Figure 4**.

The UserControl that defines the calendar page is called CalendarPage, and the XAML file is shown in **Figure 5**. A TextBlock near the top displays the month and year. This is followed by a second Grid with seven columns for the days of the week and six rows for up to six weeks or partial weeks in a month.

Unlike most UserControl derivatives, CalendarPage defines a constructor with a parameter, as shown in **Figure 6**.

The parameter is a DateTime, and the constructor uses the Month and Year properties to create a Border containing a TextBlock for each day of the month. These are each assigned Grid.Row and Grid.Column attached properties, and then added to the Grid. As you know, often months only span five weeks, and occasionally February only has four weeks, so RowDefinition objects are actually removed from the Grid if they're not needed.

UserControl derivatives normally don't have constructors with parameters because they usually form parts of larger visual trees. But CalendarPage isn't used like that. Instead, the PrintPage handler simply assigns a new instance of CalendarPage to the PageVisual property of PrintPageEventArgs. Here's the complete body of the handler, clearly illustrating how much work is being performed by CalendarPage:

```
args.PageVisual = new CalendarPage(dateTime);
args.HasMorePages = dateTime < dateTimeEnd;
dateTime = dateTime.AddMonths(1);
```

Adding a printing option to a program is so often viewed as a grueling job involving lots of code. To be able to define most of a printed page in a XAML file makes the whole thing much less frightful. ■

**CHARLES PETZOLD** *is a longtime contributing editor to* MSDN Magazine. *His new book, "Programming Windows Phone 7" (Microsoft Press, 2010), is available as a free download at bit.ly/cpebookpdf.*

**THANKS** to the following technical experts for reviewing this article: *Saied Khanahmadi and Robert Lyon*



# R.I.P., DEC

Ken Olsen died on Feb. 6, and the industry press scrambled to eulogize him. My column deadline, that dead-tree snail-mail thing, had already passed for *MSDN Magazine*'s March issue. I couldn't bump Simba's column because April Fool's Day only comes once per year.

Olsen founded Digital Equipment Corporation, universally known as DEC (pronounced like the floor of a ship) in 1957. DEC was famous for its headquarters in a refurbished woolen mill—the symbol of the "Massachusetts Miracle," new industry rising from the ashes of the old. The first computer I ever programmed and played text-based Star Trek on—it didn't have the graphics for Solitaire—was a DEC PDP-10 I met in college.

DEC was where I got my very first start teaching Windows to industry: the 16-bit SDK in C, using Charles Petzold's "Programming Windows, Second Edition" (Microsoft Press, 1990) as the textbook. The DEC students hated it: "Near pointers and far pointers? Memory segments? Instance thunks? Are you drunk/high/kidding/crazy?"

The DEC PDP and VAX minicomputers became exceedingly popular. DEC grew to be the second largest computer company in the world, after IBM. At its zenith, DEC owned a fleet of helicopters for shuttling employees among its local sites and to the airport, bypassing the area's (then) fierce traffic.

But those whom the gods would destroy, they first trap in a self-reinforcing positive feedback cycle. DEC failed to see the coming of the PC. To the company, the world needed VAXs and more VAXs. PCs were toys. Then Windows 3.1 hit, PCs became somewhat useful, and no one wanted VAXs anymore. You could see the guard changing in 1988, when Microsoft hired away Dave Cutler, the chief architect of the DEC VMS OS, to design the Windows NT kernel. I taught DEC some NT, which ran on one of its 64-bit "Alpha" RISC chips, but that never enjoyed much success. DEC shrank and died. Compaq bought the remnants in 1998. (There's an alumni association at decconnection.org, and the company's employee credit union lives on as an independent entity, dcu.org.)

I stopped by the mill recently on my way to a client. Part of it is now a self-storage facility; a place, as George Carlin said, "to keep your stuff while you're out getting more stuff." Other tenants include a preschool, and lots and lots of bats.

It's easy to take cheap shots at DEC and Olsen for not understanding what hit them, but DEC had a lot of company in its mini-blindness. None of the other then-successful minicomputer makers, often founded or staffed by DEC refugees, managed to evolve and survive. Data General. Wang. Prime. Apollo. Computervision. Gone now, all gone. *Sic transit gloria mundi*.

As George Will (another of my influences—he also writes a back-page column for some obscure rag) wrote about the revolution now bubbling in Egypt and elsewhere: "Those Americans who know which Republican will win next year's Iowa caucuses can complain about those who did not know that when a Tunisian street vendor set himself on fire, he would set a region afire. From all other Americans, forbearance would be seemly." Point well taken, if somewhat difficult to parse. Will was saying that it's hard to predict the future.

**The remains of what was once DEC headquarters.**

Ken Olsen and DEC, and many others, built a bridge from the glass house where you begged an operator to run your batch computing job and he got back to you if and when he felt like it, to one where the computer terminal sat on everyone's desktop, at their command. We couldn't have progressed from where we were to where we are now without the bridge that he built. And if he didn't cross it himself, and stood on the other side wondering, "Hey, where the heck are you guys all going?"—that doesn't change the fact that he built it when we needed it.

The world is a better place for DEC, and Olsen, having been in it. I wouldn't mind if someone said that about me when I'm gone. *Bayete*, Ken Olsen and DEC.

Next month, I'll tell you what Microsoft has to do to avoid killing itself off the same way that DEC did. ■

**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.*

*"Kindness is the language which the deaf can hear and the blind can see." ~Mark Twain*

# Relief for Japan


*GrapeCity's Headquarters is in Sendai, Japan.*

*Our colleagues there are on the ground helping the most affected.*

*Join us in the re-building effort.*

GrapeCity's Headquarters, located in Sendai, Japan, was affected by the earthquake disaster. While the office recently reopened and only suffered minor structural damage, many of the employees' families, as well the entire region's residents have been struggling to receive basic necessities.

Our colleagues at GrapeCity are on the ground helping the most affected. Please join us in the re-building effort. GrapeCity US is encouraging those who would like to help out with the relief and recovery efforts in the affected communities to contribute to the Relief for Japan fund.

Many of you who have known and worked with the GrapeCity, FarPoint and Data Dynamics employees over the years have expressed a personal desire to contribute to the earthquake and tsunami relief efforts. In response to the overwhelming requests, we have set up this online payment service.

The funds contributed will be directly used in the community relief efforts being conducted by the GrapeCity employees.

Contribute now at
**www.GCPowerTools.com/Relief**

***Thank You.***

Microsoft® tech·ed North America 2011
MAY 16-19, 2011 | ATLANTA
Visit our booth **603** at the Tech•Ed Show.

GrapeCity PowerTools
Report Analyze Excel
www.GCPowerTools.com

ACTIVEREPORTS SPREAD DATA DYNAMICS REPORTS ACTIVEANALYSIS

Dundas Dashboard

# The Dashboard Framework for Developers like you


## V2.5 Now Available

**Dundas Dashboard** was built with developers and IT staff in mind. Whether it's our open API, simple web integration or powerful scripting capabilities, technologists have all the tools and options they need for getting their dashboard projects up and running quickly and easily.

Powered by Microsoft® Silverlight™

Dundas Data Visualization

**www.dundas.com**

(416) 467-5100 • (800) 463-1492

XCEED
MULTI-TALENTED COMPONENTS

sales@xceed.com
Tel: 800.865.2626
Tel: 450.442.2626
xceed.com

# XCEED DataGrid for WPF

## The most advanced WPF data presentation control

Makes your WPF app shine by displaying your data in a rich, fluid, and high-performance datagrid. Outperforms everything on the market, handling millions of rows and thousands of columns effortlessly. Never leaves end-users waiting, supporting powerful asynchronous data virtualization for greater speed and responsiveness.

The smooth-scrolling animated Tableflow™ 2D view greatly enhances any application's user experience. This multi-talented datagrid can also display your data in true 3D with the Cardflow™ 3D view, as well as in traditional 2D with non-animated table and card views.


Rock-solid 4-year track record. Other products are either new and unproven, or have been left stagnant. Xceed DataGrid for WPF has had 45 updates in the last 4 years, which means added features requested by developers and improved stability on a nearly monthly basis.

Feature-rich. Provides far more features than any other offering: powerful grouping, hierarchical master-detail, rich printing capabilities, editors, meticulously crafted themes, filter row, column chooser, summaries, and more. 150 features in all.

Perfectly suited to large teams or complex projects. Provides the best experience on the market with remote data sources and greatly simplifies the coding for data virtualization with LINQ. Facilitates development using MVVM. Integrates seamlessly into automated user interface test procedures (Coded UI Test).

Trusted by industry leaders for mission-critical scenarios. Used by Microsoft in Visual Studio 2010!

### *Quick facts*

- Display & edit data in stunning 2D or 3D
- Highest-performance WPF datagrid
- Most adopted, most mature WPF control
- 150 features, 15 major releases in 4 years

XCEED
# DataGrid
for Silverlight

## A datagrid user experience like no other

- Lets your end-users experience fast remote data retrieval, with sources hosted on the Internet, and a completely fluid UI. Always responsive!
- Makes working with data easy for users, with lightning-fast, nested grouping with sticky group headers/footers. Filtering, sorting, and rich in-place editing also let them get a better handle on their data.
- Provides eight gorgeous themes, so your apps truly pop.

### *DataGrid quick facts*

- Beautifully rendered tabular format for Silverlight
- Advanced data virtualization for blazing speed
- Smooth-scrolling, responsive experience

XCEED
# Ultimate ListBox
for Silverlight

## Speed and beauty in a sleek format

- Gives end-users both a smooth-scrolling experience and fast remote data access, using the same pioneering background data retrieval technology as our Silverlight datagrid. A simple drop-in replacement of the stock Silverlight listbox that offers so much more!
- Lets end-users jump quickly to the info they need, thanks to convenient grouping with sticky headers/footers, filtering, sorting, and data navigation features.
- Provides six beautiful themes. Also provides an innovative “path” view, which displays items along a path in the viewport, allowing unique 3D-like presentations, in addition to the traditional vertical list layout.


XCEED
MULTI-TALENTED COMPONENTS


sales@xceed.com
Tel: 800.865.2626
Tel: 450.442.2626
xceed.com


### *ListBox quick facts*

- The streamlined format of a listbox
- Lightning-fast data virtualization technology
- Fluid, smooth-scrolling UI that’s always responsive