THE MICROSOFT JOURNAL FOR DEVELOPERS

OCTOBER 2014 VOL 29 NO 10

msdn magazine

C#

THE MICROSOFT JOURNAL FOR DEVELOPERS

OCTOBER 2014 VOL 29 NO 10

msdn magazine

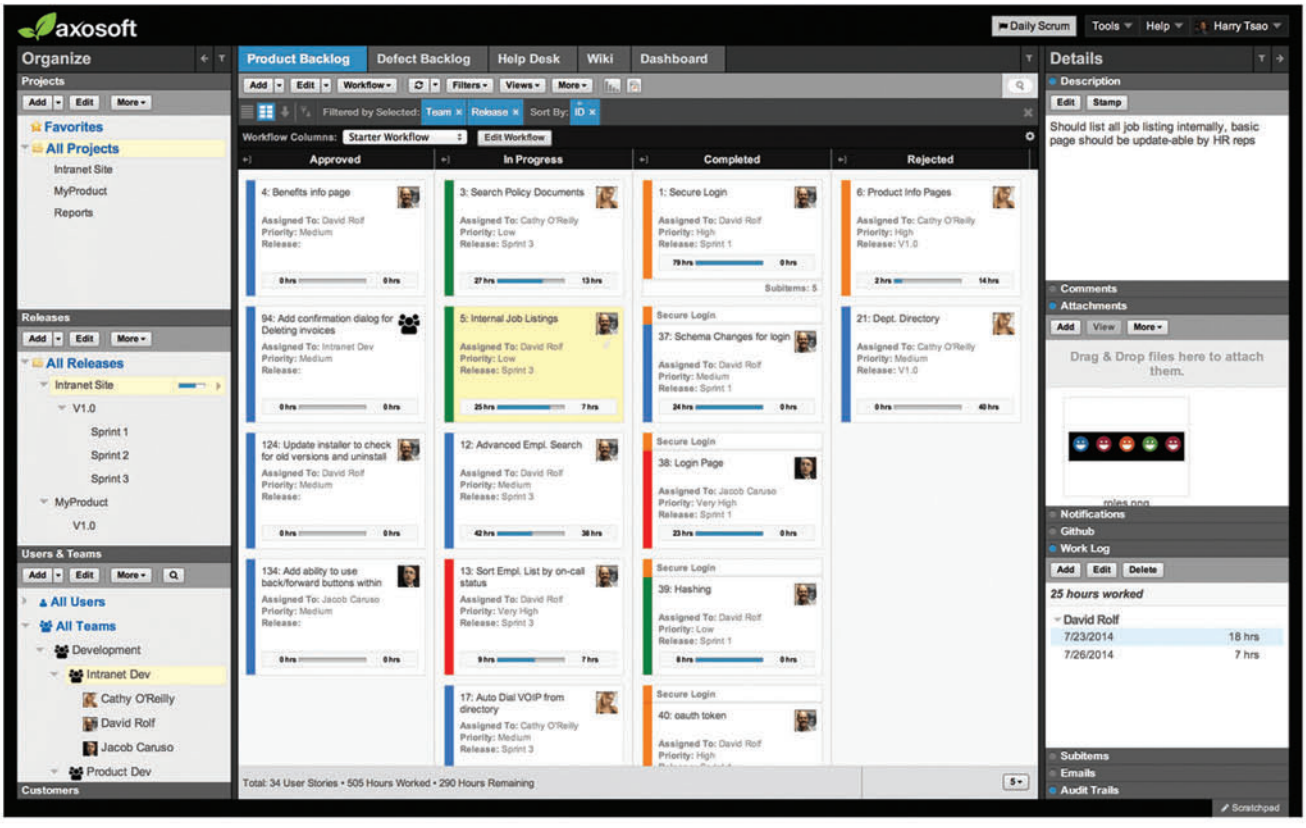
COLUMNS

Microsoft

Don't let your dev team get bogged down by complicated processes.

axosoft

Transform your team into a lean, mean Scrum machine with Axosoft!



Empower management & team members with total visibility into your development process.

Axosoft Scrum provides key project insights with a Kanban card view that clearly illustrates an item's workflow status, while burndown charts and custom reports help management identify further opportunities for improvement. Out-of-the-box the tool is configured with Scrum best practices in mind! Get started free at **Axosoft.com/MSDNscrum**.

msdn magazine

OCTOBER 2014 VOLUME 29 NUMBER 10

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

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, Jane Long, Merit Direct. Phone: 913-685-1301; E-mail: jlong@meritdirect.com; Web: www.meritdirect.com/1105

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

Microsoft

BPA WORLDWIDE

Printed in the USA

LEADTOOLS®

THE WORLD LEADER IN IMAGING SDKs

THE PREMIER **PDF** SDK

Comprehensive PDF SDK technology for today's digital world. Develop multi-platform applications with support for extracting, editing and writing text, hyperlinks, annotations, bookmarks, metadata and pages from PDF files on any desktop, tablet and mobile device.

.NET Windows API WinRT Linux iOS OS X Android HTML5/JavaScript

DOWNLOAD OUR 60 DAY EVALUATION
WWW.LEADTOOLS.COM

LEAD TECHNOLOGIES INCORPORATED

SALES@LEADTOOLS.COM
800.637.1840

MICHAEL DESMOND

Arrivals and Departures

The October issue of *MSDN Magazine* comes to you bearing both good news and bad news. I'll start with the tough news first. This issue will be the last featuring Charles Petzold as a regular columnist in *MSDN Magazine*. Petzold has signed on at Xamarin, where he is helping developers grasp the finer points of the Xamarin C# tools for cross-platform Android and iOS app development. Unfortunately, that means he no longer has time to write his monthly column for us.

Petzold has been associated with *MSDN Magazine* and its predecessor, *Microsoft Systems Journal*, going all the way back to 1986, and he's been a regular columnist here since teaming up with Juval Lowy and Matt Milner on the Foundations column in 2007. Since then, Petzold has authored several columns for us in a more-or-less uninterrupted string, including UI Frontiers, Touch and Go, and (most recently) DirectX Factor. That last column, which launched in January 2013, dove into the sometimes-arcane world of DirectX and native C++ development.

> "On average, I'm sure I spent more time programming and writing for each DirectX Factor column than any previous magazine writing."
>
> — *Charles Petzold*

"On average, I'm sure I spent more time programming and writing for each DirectX Factor column than any previous magazine writing," Petzold says. "I enjoyed the experience immensely, but I think DirectX is intrinsically a full-time job."

Now, Petzold is moving on, entering what he calls "a whole new chapter" in his life. He's been writing a book about Xamarin.Forms, an API abstraction layer for iOS, Android, and Windows Phone programming using C# and XAML. And you'll find Petzold at the Xamarin Evolve conference this month (Oct. 6-10), where he'll be speaking about Xamarin.Forms.

It's been our honor at *MSDN Magazine* to host Petzold for all these years, and we hold out hope that he'll visit our pages again sometime soon, perhaps writing a feature or two about cross-platform mobile development.

Now, for the good news: *MSDN Magazine* this month welcomes Microsoft Principal Director Keith Boyd as a member of the editorial team. Boyd heads a team of about 50 programming writers at Microsoft and is in charge of all the developer documentation in the Microsoft Cloud and Enterprise division. His arrival immediately improves the magazine's access to top-shelf developer content and ensures it can better fulfill its mission of helping working developers master the tools and techniques critical to their jobs.

"Our goal is to thoroughly modernize the content of the site and to make it as discoverable as possible, so more members of our developer community can benefit from it," Boyd says. "That means improved content presentation, better integration of the magazine content with other MSDN properties, improved discovery of related code assets, and better linking to related content and videos. We want people to come away from the magazine Web site inspired to try something new and motivated to make a bet on Microsoft."

There's a lot to be excited about as we round the corner into 2015. Boyd singles out the work Microsoft CEO Satya Nadella has done to refocus the company, and looks forward to the continued evolution of Visual Studio and Visual Studio Online. He also notes coming advancements in both Windows and Windows Phone, as well as the rapid-fire innovation coming from the Microsoft Azure team. As Boyd notes, "It's an exciting time to be a part of Microsoft."

And it's an exciting time to be at *MSDN Magazine*. We're looking forward to working with Boyd and his team to keep readers abreast of everything that's going on in the Microsoft ecosystem.

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

Data Quality Tools for Developers

A better way to build in data verification

Since 1985, Melissa Data has provided the tools developers need to enhance databases with clean, correct, and current contact data. Our powerful, yet affordable APIs and Cloud services provide maximum flexibility and ease of integration across industry-standard technologies, including .NET, Java, C, and C++. Build in a solid framework for data quality and protect your investments in data warehousing, business intelligence, and CRM.

- **Verify international addresses for over 240 countries**
- **Enhance contact data with phone numbers and geocodes**
- **Find, match, and eliminate duplicate records**
- **Sample source code for rapid application development**
- **Free trials with 120-day ROI guarantee**

Address Verification

Phone Verification

Email Verification

Geocoding

Matching/ Dedupe

Change of Address

Melissa Data.
Architecting data quality success.

MELISSA DATA®

| United States | United Kingdom | Germany | India |

DINO ESPOSITO

Source Code Readability Tips

Have you ever heard of the International Obfuscated C Code Contest? In a nutshell, it's an open contest that selects a winner from a handful of C programs that solves a problem—any problem—with an extremely obscure and obfuscated C code. You can find the source code of winning programs in previous years at ioccc.org/years.html.

The Obfuscated C Code Contest is a lighthearted way to demonstrate the importance of style and readability in programming. This column will summarize some of the most important practices you'll want to follow in order to have code that's easy to read and understand—both for your own sake and that of your colleagues.

Readability as an Attribute

In software development, maintainability is the attribute that refers to the ease with which you can modify existing code to achieve goals such as fixing a bug, housekeeping, implementing a new feature or just refactoring to some patterns. Maintainability is one of the fundamental attributes of software, according to the ISO/IEC 9126 paper. For more information on software attributes, refer to the paper at bit.ly/VCpe9q.

Code maintainability results from a variety of factors, one of which is readability. Code that's hard to read is also hard to understand. Developers who put their hands on code they don't clearly know and understand are liable to make the code even worse.

Unfortunately, readability is an extremely subjective matter. Developing an automatic tool to check and report on the readability level of the code is virtually impossible. However, even if automatic readability measurement were possible, any such tools would likely be considered highly unreliable and wouldn't be trusted by anybody. In the end, readability is a manual attribute that individual developers should check along with the rest of their code. The ability to write code that's easy to read should be part of the cultural responsibility of individual developers, extending their skill set.

Generally speaking, readability is a code attribute you can and should learn to adopt right at the beginning of your programming career and develop and improve over time. Like style and good design, readability shouldn't be reserved for experts. More important, it shouldn't be postponed to when you just have enough time for it.

A Pragmatic Approach to Readability

Producing readable code is a matter of respect for other developers. As a StackOverflow user once posted, "you should always code as if the person who ends up maintaining your code is a violent psychopath who knows where you live." You should also consider the developer who ends up maintaining your code, one day, might actually be you.

When reading other people's code, there are a couple of things that can drive you crazy. One aspect that makes it difficult to read code is data structures and algorithms with no clear goals. Another aspect is unclear strategies used in code that are difficult to determine and not well notated through comments. Here's an example:

```
// Find the smallest number in a list of integers
private int mininList(params int[] numbers)
{
  var min = Int32.MaxValue;
  for (var i = 0; i < numbers.length; i++) {
    int number = numbers[i];
    if (number < min)
      min = number;
  }
  return min;
}
```

Even though I ported the example in C# for clarity, I have to admit that this is not a piece of code that any C# developer would ever consider writing. The reason is that with C# and the Microsoft .NET Framework, you can achieve many of the same results using LINQ. I encountered similar code in a project, except that it was written in Java. At some point, I was hired to port the codebase to the .NET Framework and C#.

You could also come across code like that in a real .NET project. You can apply some readability considerations without losing your intent. The first thing that doesn't work in that function is the name. The convention is arguable. The name misses a verb and uses a mixed casing logic. Something like GetMinFromList would've probably been a better name. However, the most debatable point of the code is the private qualifier used in the name.

Any casual reader of that code can see that the function serves as a utility. In other words, it represents a potentially reusable piece of code you can call from a variety of places within the rest of the codebase. Therefore, marking it as private doesn't always make sense. However, developers know the power of the YAGNI rule—You Ain't Gonna Need It—and, reasonably, tend to not expose code that isn't strictly needed.

The author of that code could've foreseen the function as potentially reusable, but not when it was written. That's why that function was written to be easily turned into a reusable helper function, but was marked private to be visible only within the host class. This coding strategy might be hard to figure out immediately for external readers. However, it's just a decision that requires a few lines of comments to explain its motivation. If you don't add appropriate comments, you aren't being a good citizen in the coding world. Readers ultimately make sense of it, but it wastes a few minutes and, worse yet, it makes readers somewhat hostile to the author.

Practical Rules of Readability

Code readability is one of those subjects whose importance is widely recognized, but not necessarily formalized. At the same time, without some formalization, code readability is nearly an empty concept.

DOMAINS | MAIL | **HOSTING** | eCOMMERCE | SERVERS

MAXIMUM PERFORMANCE WITH MINIMAL EFFORT

New 1&1 Performance Hosting packages combine the performance of dedicated hardware **plus** the strength of our powerful hosting platform! Get the benefits of a managed hosting environment so you can focus on your web projects.

Powerful

- Up to 32 GB of dedicated RAM
- Up to 2,000 GB SATA HDD
- Software RAID 1
- 1&1 SSL Certificate
- 1&1 CDN with Railgun™
- 1&1 SiteLock

WordPress & App Platforms

- Supports over 140 popular apps including WordPress and more
- App Expert support
- Trial version available for all applications
- Security & version upgrade notifications

All Inclusive

- 1&1 Mobile Website Builder
- Latest PHP versions
- Static IP address

NEW:

GUARANTEED PERFORMANCE ON YOUR OWN SERVER!

Starting at

$59.99 per month*

~~$69.99~~

1 TRIAL 30 DAY MONEY BACK GUARANTEE

1 MONTH FLEXIBLE PAYMENT OPTIONS

1 CALL SPEAK WITH AN EXPERT 24/7

1 (877) 461-2631

1&1®

1and1.com

*Offer valid for a limited time only. The $59.99/month price is valid for the initial 1, 12, or 24 month term for the 1&1 Performance Hosting S package. Regular and renewal price is $69.99/month. $49 setup fee applies. Visit www.1and1.com for full promotional details. Program and pricing specifications and availability are subject to change without notice. 1&1 and the 1&1 logo are trademarks of 1&1 Internet, all other trademarks are the property of their respective owners.

Overall, you can approach readability with the rule of the three C's: a combined function of comments, consistency and clarity.

IDE tools are much smarter today than just a few years ago. Developers have no strict need to write help files and integrate their custom documentation in Visual Studio projects. Tool tips are everywhere and automatically created from comments. Modern IDEs make it so easy to define institutional comments, all you have to think of is the text and the IDE will do the rest. An institutional comment is the classic comment you add to methods and classes to provide a concise description of the goals. You should write these comments following platform or language standards. You should also consider these comments mandatory for any public piece of code you create.

Banning obvious comments is another fundamental step on the road to improved readability. Obvious comments just add noise and no relevant information. By definition, a comment is any explanatory text for any decision you make in the code that isn't immediately obvious. A comment should only be an insightful remark about a particular aspect of the code.

The second "C" is consistency. Every team needs to use the same guidelines for writing code. It's even better if those guidelines are used on a company-wide basis. When it comes to guidelines, many stop at the point of defining what guidelines should be and stop trying to make sense of what's right and what's wrong. I dare say that wrong or right is a secondary point compared to the importance of always doing the same thing in the same way throughout your code.

Suppose for a moment you're writing a library that does string manipulation. In several places within this library, you'll likely need to check whether a string contains a given substring. How would you do that? In the .NET Framework, as well as the Java SDK, you have at least two ways of achieving the same result. You could use the Contains or IndexOf method. Those two methods, though, serve different purposes.

The Contains method returns a Boolean answer and just tells you whether the substring is contained within a given string. The IndexOf method returns the 0-based index where the searched string is located. If there's no substring, IndexOf returns a -1. From a purely functional perspective, therefore, you can use Contains and IndexOf to achieve the same goals.

However, they give a different message to anyone reading the code, and forces the reader to take a second pass on the code to see if there's a special reason to use IndexOf instead of Contains. A single second-pass reading on a line of code isn't a problem, of course. But when it happens on an entire codebase of thousands of lines of code, it does have an impact on time and, subsequently, costs. That's the direct cost of not having highly readable code.

A sense of code consistency should be part of your innate responsibility. As a developer, you should aim to write clean code the first time without hoping to have enough time to clean it up later. As a team leader, you should enforce code consistency guidelines through check-in policies. Ideally, you shouldn't allow check-in of any code that doesn't pass a consistency test.

The latest version of ReSharper can be a considerable help in making this idea concrete. You can use these free command-line tools—a standalone set of tools—to integrate forms of code-quality analysis right into your continuous integration (CI) or version control system. These command-line tools can perform offline code inspections. This is the same set of code inspections you can perform live within Visual Studio with ReSharper installed to catch duplicates in your code. Depending on your CI customization features, you might need to wrap up command-line tools in an ad hoc component. For more information on ReSharper, check out bit.ly/1avsZ2R.

The third and final "C" of code readability is clarity. Your code is clear if you style it in a way that it reads well and easily. This includes appropriate grouping and nesting. In general, IF statements add a lot of noise to the code. Sometimes you can't avoid conditional statements—a pillar of programming languages—but trying to limit the number of IF statements keeps nesting under control and makes code easier to read. You could also use an IF… ELSE … IF … ELSE structure instead of nested IF statements.

Some tasks may require a few lines of code and it might be hard or just inappropriate to do an "Extract Method" refactoring. In this case, it's good to keep these lines in blocks separated by blank lines. It doesn't change the code substance, but keeps it easier to read. Finally, if you're looking for inspiration on how to style your source code, have a look at some open source projects.

Short Is Better

Longer lines make it hard for human eyes to read. That's why newspapers and magazines print their text in columns. When it comes to your code, you should do the same and limit both the horizontal length of lines and the vertical scrolling of the methods. What's the ideal length of a method's body? Generally, 30 lines should be a maximum level that triggers an alarm bell and suggests you consider refactoring. Finally, a good organization of project folders and matching between folders and namespaces often reflects a good organization of individual code elements.

When you raise the theme of readability and clean code, you're often exposed to one common objection—writing clean code is hard and takes a lot of time. You should try to neutralize this viewpoint and there are two ways you can. One is to use a code assistant tool that helps write clean code by suggesting refactorings, inspecting your code for bad patterns, and checking for dead or duplicated code. If you can make all these features accessible, you really have no more excuses for not writing cleaner and readable code. Code assistant tools are offered from major vendors. Pick any, but pick one.

The other is to emphasize self-improvement and the attitude of your individual developers to write cleaner code as a matter of general practice. Experienced developers do this well. The ability to know roughly how far away you are from the final version of your code, and then truly clean it up, is a key characteristic that separates experienced developers from the less experienced. ■

Dino Esposito *is the co-author of "Microsoft .NET: Architecting Applications for the Enterprise" (Microsoft Press, 2014) and "Programming ASP.NET MVC 5" (Microsoft Press, 2014). A technical evangelist for the .NET Framework and Android platforms at JetBrains and frequent speaker at industry events worldwide, Esposito shares his vision of software at software2cents.wordpress.com and on Twitter at twitter.com/despos.*

Thanks to the following Microsoft technical expert for reviewing this article: *James McCaffrey*

ASPOSE.TOTAL

Powerful APIs which enable developers to harness the complexity of file format processing within their apps.

Your File Format APIs

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

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

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

Aspose.Slides
PPT, PPTX, POT, POTX, XPS, HTML, PNG, PDF & other formats.

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

Aspose.BarCode
JPG, PNG, BMP, GIF, TIFF, WMF, ICON & other image formats.

Aspose.Imaging
PDF, BMP, JPG, GIF, TIFF, PNG, PSD & other image formats.

Aspose.Tasks
XML, MPP, SVG, PDF, TIFF, PNG, CSV, MPT & other formats.

Aspose.Diagram
VSD, VSDX, VSS, VST, VSX & other formats.

Aspose.Note
ONE, PNG, JPG, BMP, GIF & PDF.

... and more!

100% Standalone - No Office Automation

.NET Libraries | Java Libraries | Cloud APIs | Android Libraries

Scan for a 20% saving!

ASPOSE
Your File Format APIs

US: +1 888 277 6734
sales@aspose.com

EU: +44 141 416 1112
sales.europe@aspose.com

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

GROUPDOCS.TOTAL

Professional APIs that allow developers to empower their apps with document collaboration capabilities.

Your Document Collaboration APIs

GroupDocs.Viewer

Native-text, high-fidelity HTML5 document viewer with support for over 49 file formats.

GroupDocs.Signature

Electronic signature API that gives your apps legally binding e-signature capabilities.

GroupDocs.Conversion

Universal document converter for fast conversion between more than 49 file formats.

GroupDocs.Annotation

A powerful API that lets developers annotate Microsoft Office, PDF and other documents within their own apps.

GroupDocs.Assembly

Incorporates data entered by users through online forms Into both Microsoft Office and PDF documents.

GroupDocs.Comparison

A diff view API that allows end users to quickly find differences between two revisions of a document.

100% Standalone - No Office Automation

.NET Libraries | Java Libraries | Cloud APIs | Cloud Apps

SALES INQUIRIES: +1 214 329 9760

sales@groupdocs.com

GROUPDOCS
Your Document Collaboration APIs

www.groupdocs.com

JULIE LERMAN

A Pattern for Sharing Data Across Domain-Driven Design Bounded Contexts

For my entire programming life, reusable code and reusable data have been a driving objective. So, when I began learning about Domain-Driven Design (DDD), I struggled with its guidance to enforce such separation across its bounded contexts that the result might be duplicated code and even duplicated data. I had a small fit when some of the best minds in the DDD world attempted an intervention to help me see the potential pitfalls of my old ways. Finally, Eric Evans explained that one had to choose where to pay the price of complexity. Because DDD is about reducing complexity in the software, the outcome is that you pay a price with respect to maintaining duplicate models and possibly duplicate data.

In this column, I've written about DDD concepts and how they align with the data-driven experience, first in my January 2013 column, "Shrink EF Model with DDD Bounded Contexts" (bit.ly/1isloGE), and then in a three-part series called "Coding for Domain-Driven Design: Tips for Data-Focused Devs," which begins at bit.ly/XyCNrU. In the first of this series you can find a section titled "Shared Data Can Be a Curse in Complex Systems." Take a look at it for some additional insight as to why the approach I'll demonstrate here is useful.

For my entire programming life, reusable code and reusable data have been a driving objective.

I've been asked numerous times how, exactly, you can share data between bounded contexts if you follow one of the more extreme DDD patterns, which is to have each bounded context tied to its own database. Steve Smith and I talked about this in our Domain-Driven Design Fundamentals course on Pluralsight.com (bitly.com/PS-DDD), but we didn't actually implement it, as doing so is a bit more advanced than the focus of that particular course.

There are a number of different ways to leverage common data across bounded contexts. In this column I'm going to focus on one scenario in particular: mirroring data from one system to another where the first system is designed for editing that data and the second just needs read-only access to a bit of that data.

Code download available at msdn.microsoft.com/magazine/msdnmag1014.

I'll first lay out the basic pattern, then add some additional detail. The implementation involves a number of working parts, including Inversion of Control (IoC) containers and message queues. If you're familiar with these tools, the implementation should be easier to comprehend. I won't go into great detail about the IoC and queue implementation, but you'll be able to see and debug it in the download sample that goes with this article.

The Sample Scenario: Sharing a Customer List

I chose a very simple scenario to demonstrate this pattern. One system is dedicated to customer service. Here, users can maintain customer data, along with plenty of other data. This system interacts with a data storage mechanism, but that isn't important to the sample. The second system is designed for taking orders. In that system, users need access to customers, but really only to identify the customer making the order. That means this bounded context needs just a read-only list of customer names and identifiers. Therefore, the database connected to the second system needs an up-to-date list of customer names and IDs based on the customers that are maintained in the first system. The approach I chose to accomplish this is to mirror in the second system those two pieces of data for each customer that's maintained in the first system.

Mirroring Data: High Level

At a very high level, the solution I'll apply is that every time a customer is inserted into System A, the ID and name of that customer should be added to System B's data store. If I change an existing customer's name in System A, then System B needs to have the correct name as well, so a name change should cause an update in the System B data store. This domain doesn't delete data, but a future enhancement could be to remove inactive customers from the System B data store. I won't bother with that in this implementation.

So, from the previous paragraph, I care about only two events in System A to which I need to respond:

- A customer was inserted
- An existing customer's name was updated

In a connected system, System B could expose methods for System A to call, such as InsertCustomer or UpdateCustomerName. Or System A could raise events, such as CustomerCreated and CustomerNameUpdated, for other systems, including System B, to catch and respond to.

In response to each event, System B needs to do something in its database.

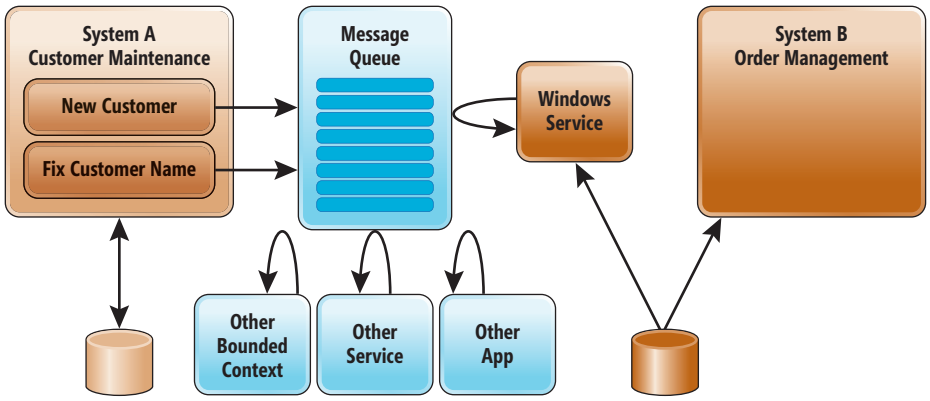


Figure 1 **A Message Queue Allows Unacquainted Systems to Share Messages, in This Case to Update the System B Database**

Because these systems are disconnected, a better approach is to employ a publish-subscribe pattern. System A will publish one or more events to some type of operator. And one or more systems then subscribe to that same operator, waiting for particular events and performing their own actions in response to those events.

Publish-subscribe aligns with the principles of DDD that require these two systems be unaware of each other and, therefore, not talk directly to one another. So, instead, I'll use a concept called an anti-corruption layer. Each system will communicate through the operator that will shuffle messages between the two.

This operator is a message queue. System A will send messages to the queue. System B will retrieve messages from the queue. In my example, I'll have only one subscriber—System B—but it's possible to have many subscribers.

What's in the Event Message?

When the event being published is the CustomerCreated event, System A will send a message that says, "A customer has been inserted. Here's the customer's identity and name." That's the full message except that it's represented in data, not in nice English sentences. The interesting part about publishing an event to a message queue is that the publisher doesn't care what systems are retrieving the message or what they'll do in response.

System B will respond to that message by inserting or updating the customer in its database. In reality, System B doesn't even perform this task; I'll let a service do the job. Moreover, I'll let the database determine how an update is to be performed. In this case, the System B database will execute a customer "update" using a stored procedure whose logic deletes the original customer record and inserts a new one. Because I'll be using GUIDs as identity keys, the identity of the customer will be maintained properly. I don't want to worry about database-generated keys in my DDD software. Pre-created GUIDs vs. database-incremented keys is a sticky topic. You'll need to define your logic to align with your company's database practices.

In the end, System B, the ordering system, will have a complete list of customers it can use. Further along in the workflow, if the ordering system needs more information about a particular customer, for example credit card information or current shipping address, I can leverage other mechanisms, such as calling a service, to retrieve that data as needed. I won't be addressing that workflow here, however.

Communicating with the Message Queue

A system that allows you to communicate messages in an asynchronous way is called an event bus. An event bus contains the infrastructure to store messages and provide them to whoever needs to retrieve them. It also provides an API for interacting with it. I'll focus on one particular implementation that I found to be an easy way to get started with this style of communication: a message queue. There are a number of message queues from which to choose. In the DDD Fundamentals course on Pluralsight.com, Smith and I chose to use the SQL Server Service Broker as our message queue. Because we both work with SQL Server, it was simple for us to set up and we just needed to write SQL to push messages to the queue and retrieve them.

In writing this article, I decided it was time for me to learn to use one of the more popular open source message queues, RabbitMQ. This meant installing the RabbitMQ server (and Erlang!) onto my computer, as well as pulling in the RabbitMQ .NET Client so I could easily code against it in my application. You can learn more about RabbitMQ at rabbitmq.com. I also found the RabbitMQ for .NET Developers courses on Pluralsight.com to be very helpful.

System A, therefore, has a mechanism for sending messages to the RabbitMQ server. But System B, the Order system, has nothing

Figure 2 **The Customer Class in the Customer Maintenance-Bounded Context**

```
public static Customer Create(string name, string source) {
   return new Customer(name, source);
  }
  private Customer(string name, string source){
    Id = Guid.NewGuid();
    Name = name;
    InitialDate = DateTime.UtcNow;
    ModifiedDate = DateTime.UtcNow;
    Source = source;
    PublishEvent (true);
  }

  public Guid Id { get; private set; }
  public string Name { get; private set; }
  public DateTime InitialDate { get; private set; }
  public DateTime ModifiedDate { get; private set; }
  public String Source { get; private set; }


  public void FixName(string newName){
    Name = newName;
    ModifiedDate = DateTime.UtcNow;
    PublishEvent (false);
  }

  private void PublishEvent(bool isNew){
    var dto = CustomerDto.Create(Id, Name);
    DomainEvents.Raise(new CustomerUpdatedEvent(dto, isNew));
 }}
```

to do with any of this interaction. System B simply expects the customer list to be in the database and doesn't care how it gets there. A separate small Windows service will handle checking the RabbitMQ message queue for messages and updating the Order system's database accordingly. **Figure 1** shows a visual workflow of the entire process.

Sending Messages to the Queue

I'll start with the Customer class in System A, shown in **Figure 2**. For the sake of a simple example, it comprises only a few properties—ID, Name, the source of the customer and some logging dates. Following DDD patterns, the object has built-in constraints to prevent random editing. You create a new customer using the Create factory method. If you need to fix the name, you use the FixName method.

Notice that the constructor and the FixName method both call the PublishEvent method, which, in turn, creates a simple CustomerDto (which has only an Id and Name property) and then uses the DomainEvents class from Udi Dahan's 2009 *MSDN Magazine* article, "Employing the Domain Model Pattern" (msdn.microsoft.com/magazine/ee236415) to raise a new CustomerUpdatedEvent (see **Figure 3**). In my example, I'm publishing the event in response to simple actions. In a real implementation, you might prefer to publish these events after the data has been successfully persisted into the System A database.

A CustomerUpdatedEvent wraps all I need to know about this event: the CustomerDto along with a flag that indicates whether the customer is new. There's also metadata that will be needed by a generic event handler.

The CustomerUpdatedEvent can then be handled by one or more handlers I define in my application. I've defined only one handler, a service called CustomerUpdatedService:

```
public class CustomerUpdatedService : IHandle<CustomerUpdatedEvent>
{
  private readonly IMessagePublisher _messagePublisher;

  public CustomerUpdatedService(IMessagePublisher messagePublisher){
    _messagePublisher = messagePublisher;
  }

  public void Handle(CustomerUpdatedEvent customerUpdatedEvent){
    _messagePublisher.Publish(customerUpdatedEvent);
}}
```

Figure 3 **A Class That Encapsulates an Event When a Customer Is Updated**

```
public class CustomerUpdatedEvent : IApplicationEvent{
  public CustomerUpdatedEvent(CustomerDto customer, bool isNew) : this(){
    Customer = customer;
    IsNew = isNew;
  }

  public CustomerUpdatedEvent()
  {
    DateTimeEventOccurred = DateTime.Now;
  }

  public CustomerDto Customer { get; private set; }
  public bool IsNew { get; private set; }
  public DateTime DateTimeEventOccurred { get; set; }

  public string EventType{
    get { return "CustomerUpdatedEvent"; }
}}
```

The service will handle all CustomerUpdatedEvent instances my code raises by using the specified message publisher to publish the event. I haven't specified the publisher here; I've only referenced an abstraction, IMessagePublisher. I'm employing an IoC pattern that lets me loosely couple my logic. I'm a fickle gal. Today I may want one message publisher. Tomorrow, I might prefer another. In the background, I used StructureMap (structuremap.net), a popular tool among .NET developers for managing IoC in .NET applications. StructureMap lets me indicate where to find the classes that handle events raised by DomainEvents.Raise. StructureMap's author, Jeremy Miller, wrote an excellent series in *MSDN Magazine* called "Patterns in Practice" that's relevant to the patterns applied in this sample (bit.ly/1ltTgTw). With StructureMap, I configured my application to know that when it sees IMessagePublisher, it should use the concrete class, RabbitMQMessagePublisher, whose logic is shown here:

```
public class RabbitMqMessagePublisher : IMessagePublisher{
  public void Publish(Shared.Interfaces.IApplicationEvent applicationEvent) {
    var factory = new ConnectionFactory();

    IConnection conn = factory.CreateConnection();
    using (IModel channel = conn.CreateModel()) {
      [code to define the RabbitMQ channel]
      string json = JsonConvert.SerializeObject(applicationEvent, Formatting.None);
      byte[] messageBodyBytes = System.Text.Encoding.UTF8.GetBytes(json);
      channel.BasicPublish("CustomerUpdate", "", props, messageBodyBytes);
 }}}
```

Note that I've removed a number of lines of code specific to configuring RabbitMQ. You can see the full listing in the download (msdn.microsoft.com/magazine/msdnmag1014).

The meat of this method is that it publishes a JSON representation of the event object into the queue. Here's what that string looks like when I've added a new customer named Julie Lerman:

```
{
"Customer":
  {"CustomerId":"a9c8b56f-6112-42da-9411-511b1a05d814",
    "ClientName":"Julie Lerman"},
"IsNew":true,
"DateTimeEventOccurred":"2014-07-22T13:46:09.6661355-04:00",
"EventType":"CustomerUpdatedEvent"
}
```

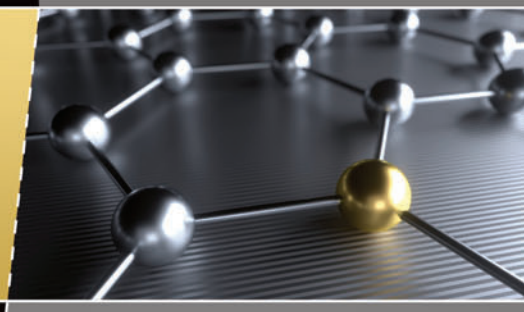
When this message has been published, the Customer Maintenance system's involvement is complete.

In my sample application, I use a set of tests to cause messages to be published to the queue, as shown in **Figure 4**. Rather than build tests that will check the queue when they're finished, I simply browse to the RabbitMQ Manager on my computer and use its tooling. Notice in the test constructor I initialize a class called IoC. This is where I've configured StructureMap to wire up the IMessagePublisher and my event handlers.

Retrieving the Message and Updating the Order System's Database

The message sits on the RabbitMQ server waiting to be retrieved. And that task is performed by a Windows service that runs continuously, periodically polling the queue for new messages. When it sees a message, the service retrieves and handles it. The message can also be handled by other subscribers as they come along. For the sake of this sample, I created a simple console application, rather than a service. This allows me to easily run and debug the "service" from Visual Studio while learning. For

Switch to Amyuni PDF



Create & Edit PDFs in .Net - ActiveX - WinRT

- Edit, process and print PDF 1.7 documents
- Create, fill-out and annotate PDF forms
- Fast and lightweight 32- and 64-bit components for .Net and ActiveX/COM
- New WinRT Component enables publishing C#, C++CX or Javascript apps to Windows Store
- New Postscript/EPS to PDF conversion module

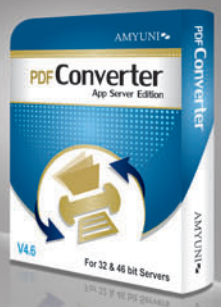
Complete Suite of Accurate PDF Components

- All your PDF processing, conversion and editing in a single package
- Combines Amyuni PDF Converter and PDF Creator for easy licensing, integration and deployment
- Includes our Microsoft WHQL certified PDF Converter printer driver
- Export PDF documents into other formats such as JPeg, PNG, XAML or HTML5
- Import and Export XPS files using any programming environment

Advanced HTML to PDF & XAML

- Direct conversion of HTML files into PDF and XAML without the use of a web browser or a printer driver
- Easy Integration and deployment within developer's applications
- WebkitPDF is based on the Webkit Open Source library and Amyuni PDF Creator

High Performance PDF Printer For Desktops and Servers



- Our high-performance printer driver optimized for Web, Application and Print Servers. Print to PDF in a fraction of the time needed with other tools. WHQL tested for Windows 32 and 64-bit including Windows Server 2012 R2 and Windows 8.1
- Standard PDF features included with a number of unique features. Interface with any .Net or ActiveX programming language
- Easy licensing and deployment to fit system administrator's requirements

CERTIFIED FOR Windows Server 2008 · Windows 7 · Windows 8 · Windows Server 2012 Certified

AMYUNI

All development tools available at

www.amyuni.com

USA and Canada
Toll Free: 1866 926 9864
Support: 514 868 9227
sales@amyuni.com

Europe
UK: 0800-015-4682
Germany: 0800-183-0923
France: 0800-911-248

the next iteration, I might check out Microsoft Azure WebJobs (bit.ly/1l3PTYH), rather than tangling with a Windows service or using my console application hack.

The service employs similar patterns of raising events with Dahan's DomainEvents class, responding to events in a handler class and initializing an IoC class that uses StructureMap to locate the event handlers.

The service listens to RabbitMQ for messages using the RabbitMQ .NET Client Subscription class. You can see the logic for this in the following Poll method, where the _subscription object keeps listening for messages. Every time a message is retrieved, it deserializes the JSON I stored in the queue back into a CustomerUpdatedEvent and then raises the event:

```
private void Poll() {
  while (Enabled) {
    var deliveryArgs = _subscription.Next();
    var message = Encoding.Default.GetString(deliveryArgs.Body);
    var customerUpdatedEvent =
      JsonConvert.DeserializeObject<CustomerUpdatedEvent>(message);
    DomainEvents.Raise(customerUpdatedEvent);
}}
```

The service contains a single class, Customer:

```
public class Customer{
  public Guid CustomerId { get; set; }
  public string ClientName { get; set; }
}
```

When the CustomerUpdatedEvent is deserialized, its Customer property—originally populated by a CustomerDto in the Customer Management system—deserializes to this service's Customer object.

What happens to the raised event is the most interesting part of the service. Here's the class, CustomerUpdatedHandler, that handles the event:

```
public class CustomerUpdatedHandler : IHandle<CustomerUpdatedEvent>{
  public void Handle(CustomerUpdatedEvent customerUpdatedEvent){
    var customer = customerUpdatedEvent.Customer;
    using (var repo = new SimpleRepo()){
      if (customerUpdatedEvent.IsNew){
        repo.InsertCustomer(customer);
      }
      else{
        repo.UpdateCustomer(customer);
 }}}}
```

This service uses Entity Framework (EF) to interact with the database. **Figure 5** shows the relevant interaction is encapsulated in two methods—InsertCustomer and UpdateCustomer—in a simple repository. If the event's IsNew property is true, the service calls the repository's InsertCustomer method. Otherwise, it calls the UpdateCustomer method.

Those methods perform the relevant logic using an EF DbContext. For an insert, it adds the customer and then calls SaveChanges. EF will perform the database insert command. For an update, it will send the CustomerID and CustomerName to a stored procedure, which uses whatever logic I—or my trusted DBA—has defined to perform the update.

Therefore, the service performs the necessary work in the database to make sure the Customers list in the Ordering system always has an up-to-date roster of customers as maintained in the Customer Maintenance system.

Yes, This Is a Lot of Layers and Puzzle Pieces!

Because I used a simple sample to demonstrate this workflow, you might be thinking the solution is an incredible amount of overkill. But remember, the point is that this is how to orchestrate the solution when you're using DDD practices to solve complex software problems. When focusing on the domain of customer maintenance, I don't care about other systems. By using abstractions with the IoC, handlers and message queues, I can to satisfy needs of external systems without muddying up the domain itself. The Customer class simply raises an event. For this demo, this is the easiest place to ensure the workflow makes sense to readers, but it might already be too muddy for your domain. You can certainly raise the event from another place in your application, perhaps from a repository just as it is about to push changes into its own data store.

The sample solution download for this column does employ RabbitMQ and that requires installing its lightweight, open source server on your computer. I've included references in the download's ReadMe file. I'll also post a short video on my blog at thedatafarm.com, so you can see me stepping through the code, inspecting the RabbitMQ Manager and the database to see the results. ■

Figure 4 **Publishing to RabbitMq in Tests**

```
[TestClass]
public class PublishToRabbitMqTests
{
  public PublishToRabbitMqTests()
  {IoC.Initialize();
  }
  [TestMethod]
  public void CanInsertNewCustomer()
  {
    var customer = Customer.Create("Julie Lerman", "Friend Referral");
    Assert.Inconclusive("Check RabbitMQ Manager for a message re this event");
  }
  [TestMethod]
  public void CanUpdateCustomer() {
    var customer = Customer.Create("Julie Lerman", "Friend Referral");
    customer.FixName("Sampson");
    Assert.Inconclusive("Check RabbitMQ Manager for 2 messages re these events");
}}
```

Figure 5 **The InsertCustomer and UpdateCustomer Methods**

```
public void InsertCustomer(Customer customer){
  using (var context = new CustomersContext()){
    context.Customers.Add(customer);
    context.SaveChanges();
}}
public void UpdateCustomer(Customer customer){
  using (var context = new CustomersContext()){
    var pId = new SqlParameter("@Id", customer.CustomerId);
    var pName = new SqlParameter("@Name", customer.ClientName);
    context.Database.ExecuteSqlCommand
      ("exec ReplaceCustomer {0}, {1}", customer.CustomerId, customer.ClientName);
}}
```

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.NET topics at user groups and conferences around the world. She blogs at thedatafarm.com/blog and is the author of "Programming Entity Framework" (2010), as well as a Code First edition (2011) and a DbContext edition (2012), all from O'Reilly Media. Follow her on Twitter at twitter.com/julielerman and see her Pluralsight courses at juliel.me/PS-Videos.*

Thanks to the following Microsoft technical expert for reviewing this article: *Cesar de la Torre*

Desktop. Web. Mobile. Your next great app starts here.

From interactive Desktop applications, to immersive Web and Mobile solutions, development tools built to meet your needs today and ensure your continued success tomorrow.

Download your free 30-day trial today and Experience the DevExpress Difference.

www.devexpress.com/try

DevExpress®

WIN ASP WPF SL VCL XAF CR

The New and Improved C# 6.0

Mark Michaelis

Although C# 6.0 isn't yet complete, it's at a point now where the features are close to being finalized. There have been a number of changes and improvements made to C# 6.0 in the CTP3 release of the next version of Visual Studio, code-named "14," since the May 2014 article, "A C# 6.0 Language Preview" (msdn.microsoft.com/magazine/dn683793.aspx).

> There have been a number of changes and improvements to C# 6.0.

In this article, I'll introduce the new features and provide an update on the features discussed back in May. I'll also maintain a comprehensive up-to-date blog describing updates to each C# 6.0 feature. Check it out at itl.tc/csharp6. Many of these examples are from the next edition of my book, "Essential C# 6.0" (Addison-Wesley Professional).

This article discusses:

- New features coming in C# 6.0
- Existing features that have been enhanced or improved
- New techniques you can apply in your coding

Technologies discussed:

C# 6.0 (and older versions), Microsoft .NET Framework

Null-Conditional Operator

Even the newest .NET developers are likely familiar with the NullReferenceException. This is an exception that almost always indicates a bug because the developer didn't perform sufficient null checking before invoking a member on a (null) object. Consider this example:

```
public static string Truncate(string value, int length)
{
  string result = value;
  if (value != null) // Skip empty string check for elucidation
  {
    result = value.Substring(0, Math.Min(value.Length, length));
  }
  return result;
}
```

If it wasn't for the check for null, the method would throw a NullReferenceException. Although it's simple, having to check the string parameter for null is somewhat verbose. Often, that verbose approach is likely unnecessary given the frequency of the comparison. C# 6.0 includes a new null-conditional operator that helps you write these checks more succinctly:

```
public static string Truncate(string value, int length)
{
  return value?.Substring(0, Math.Min(value.Length, length));
}
[TestMethod]
public void Truncate_WithNull_ReturnsNull()
{
  Assert.AreEqual<string>(null, Truncate(null, 42));
}
```

As the Truncate_WithNull_ReturnsNull method demonstrates, if in fact the value of the object is null, the null-conditional operator will return null. This begs the question of what happens when the null-conditional operator appears within a call chain, as in the next example:

```
public static string AdjustWidth(string value, int length)
{
  return value?.Substring(0, Math.Min(value.Length, length)).PadRight(length);
}

[TestMethod]
public void AdjustWidth_GivenInigoMontoya42_ReturnsInigoMontoyaExtended()
{
  Assert.AreEqual<int>(42, AdjustWidth("Inigo Montoya", 42).Length);
}
```

Even though Substring is called via the null-conditional operator, and a null value?.Substring could seemingly return null, the language behavior does what you would want. It short-circuits the call to PadRight, and immediately returns null, avoiding the programming error that would otherwise result in a NullReferenceException. This is a concept known as null-propagation.

The null-conditional operator conditionally checks for null before invoking the target method and any additional method within the call chain. Potentially, this could yield a surprising result such as in the statement text?.Length.GetType.

If the null-conditional returns null when the invocation target is null, what's the resulting data type for an invocation of a member that returns a value type—given a value type can't be null? For example, the data type returned from value?. Length can't simply be int. The answer, of course, is a nullable (int?). In fact, an attempt to assign the result simply to an int will produce a compile error:

```
int length = text?.Length; // Compile Error: Cannot implicitly convert
type 'int?' to 'int'
```

The null-conditional has two syntax forms. First, is the question mark prior to the dot operator (?.). The second is to use the question mark in combination with the index operator. For example, given a collection, instead of checking for null explicitly before indexing into the collection, you can do so using the null conditional operator:

```
public static IEnumerable<T> GetValueTypeItems<T>(
  IList<T> collection, params int[] indexes)
  where T : struct
{
  foreach (int index in indexes)
  {
    T? item = collection?[index];
    if (item != null) yield return (T)item;
  }
}
```

This example uses the null-conditional index form of the operator ?[…], causing indexing into collection only to occur if collection isn't null. With this form of the null-conditional operator, the T? item = collection?[index] statement is behaviorally equivalent to:

```
T? item = (collection != null) ? collection[index] : null.
```

Note that the null-conditional operator can only retrieve items. It won't work to assign an item. What would that mean, given a null collection, anyway?

> C# developers have wondered if [invoking delegates] would be improved for the last four releases. It's finally going to happen.

Note the implicit ambiguity when using ?[…] on a reference type. Because reference types can be null, a null result from the ?[…] operator is ambiguous about whether the collection was null or the element itself was, in fact, null.

One particularly useful application of the null-conditional operator resolves an idiosyncrasy of C# that has existed since C# 1.0—checking for null before invoking a delegate. Consider the C# 2.0 code in **Figure 1**.

Leveraging the null-conditional operator, the entire set implementation is reduced to simply:

```
OnTemperatureChanged?.Invoke(this, value)
```

All you need now is a call to Invoke prefixed by a null-conditional operator. You no longer need to assign the delegate instance to a local variable in order to be thread safe or even to explicitly check the value for null before invoking the delegate.

C# developers have wondered if this would be improved for the last four releases. It's finally going to happen. This feature alone will change the way you invoke delegates.

Another common pattern where the null-conditional operator could be prevalent is in combination with the coalesce operator. Instead of checking for null on linesOfCode before invoking Length, you can write an item count algorithm as follows:

```
List<string> linesOfCode = ParseSourceCodeFile("Program.cs");
return linesOfCode?.Count ?? 0;
```

Figure 1 **Checking for Null Before Invoking a Delegate**

```
class Theremostat
{
  event EventHandler<float> OnTemperatureChanged;

  private int _Temperature;
  public int Temperature
  {
    get
    {
      return _Temperature;
    }
    set
    {

      // If there are any subscribers, then
      // notify them of changes in temperature

      EventHandler<float> localOnChanged =
        OnTemperatureChanged;
      if (localOnChanged != null)
      {
        _Temperature = value;
        // Call subscribers
        localOnChanged(this, value);
      }

    }
  }
}
```

Figure 2 **Extracting the Parameter Name with a Nameof Expression**

```
void ThrowArgumentNullExceptionUsingNameOf(string param1)
{
  throw new ArgumentNullException(nameof(param1));
}

[TestMethod]
public void NameOf_UsingNameofExpressionInArgumentNullException()
{
  try
  {
    ThrowArgumentNullExceptionUsingNameOf("data");
    Assert.Fail("This code should not be reached");
  }
  catch (ArgumentNullException exception)
  {
    Assert.AreEqual<string>("param1", exception.ParamName);
}
```

In this case, any empty collection (no items) and a null collection are both normalized to return the same count. In summary, the null-conditional operator will:

- Return null if the operand is null
- Short-circuit additional invocations in the call chain if the operand is null
- Return a nullable type (System.Nullable<T>) if the target member returns a value type
- Support delegate invocation in a thread safe manner
- Is available as both a member operator (?.) and an index operator (?[…])

Auto-Property Initializers

Any .NET developer who has ever properly implemented a struct has undoubtedly been bothered by how much syntax it takes to make the type immutable (as .NET standards suggest it should be). At issue is the fact that a read-only property should have:

1. A read-only-defined backing field
2. Initialization of the backing field from within the constructor
3. Explicit implementation of the property (rather than using an auto-property)

Figure 3 **Retrieving Other Program Elements**

```
namespace CSharp6.Tests
{
  [TestClass]
  public class NameofTests
  {
    [TestMethod]
    public void Nameof_ExtractsName()
    {
      Assert.AreEqual<string>("NameofTests", nameof(NameofTests));
      Assert.AreEqual<string>("TestMethodAttribute",
        nameof(TestMethodAttribute));
      Assert.AreEqual<string>("TestMethodAttribute",
        nameof(
        Microsoft.VisualStudio.TestTools.UnitTesting.TestMethodAttribute));
      Assert.AreEqual<string>("Nameof_ExtractsName",
        string.Format("{0}", nameof(Nameof_ExtractsName)));
      Assert.AreEqual<string>("Nameof_ExtractsName",
        string.Format("{0}", nameof(
        CSharp6.Tests.NameofTests.Nameof_ExtractsName)));
    }
  }
}
```

4. An explicit getter implementation that returns the backing field

All of this is just to "properly" implement an immutable property. This behavior is then repeated for all properties on the type. So doing the right thing requires significantly more effort than the brittle approach. C# 6.0 comes to the rescue with a new feature called auto-property initializers (CTP3 also includes support for initialization expressions). The auto-property initializer allows assignment of properties directly within their declaration. For read-only properties, it takes care of all the ceremony required to ensure the property is immutable. Consider, for example, the FingerPrint class in this example:

```
public class FingerPrint
{
  public DateTime TimeStamp { get; } = DateTime.UtcNow;
  public string User { get; } =
    System.Security.Principal.WindowsPrincipal.Current.Identity.Name;
  public string Process { get; } =
    System.Diagnostics.Process.GetCurrentProcess().ProcessName;
}
```

As the code shows, property initializers allow for assigning the property an initial value as part of the property declaration. The property can be read-only (only a getter) or read/write (both setter and getter). When it's read-only, the underlying backing field is automatically declared with the read-only modifier. This ensures that it's immutable following initialization.

> Doing the right thing requires significantly more effort than the brittle approach.

Initializers can be any expression. For example, by leveraging the conditional operator, you can default the initialization value:

```
public string Config { get; } = string.IsNullOrWhiteSpace(
  string connectionString =
    (string)Properties.Settings.Default.Context?["connectionString"])?
  connectionString : "<none>";
```

In this example, notice the use of declaration expression (see itl.tc/?p=4040) as discussed in the previous article. If you need more than an expression, you could refactor the initialization into a static method and invoke that.

Nameof Expressions

Another addition introduced in the CTP3 release is support for nameof expressions. There are several occasions when you'll need to use "magic strings" within your code. Such "magic strings" are normal C# strings that map to program elements within your code. For example, when throwing an ArgumentNullException, you'd use a string for the name of the corresponding parameter that was invalid. Unfortunately, these magic strings had no compile time validation and any program element changes (such as renaming the parameter) wouldn't automatically update the magic string, resulting in an inconsistency that was never caught by the compiler.

On other occasions, such as when raising OnPropertyChanged events, you can avoid the magic string via tree expression gymnastics

that extract the name. It's perhaps a little more irritating given the operation's simplicity, which is just identifying the program element name. In both cases, the solution was less than ideal.

To address this idiosyncrasy, C# 6.0 provides access to a "program element" name, whether it's a class name, method name, parameter name or particular attribute name (perhaps when using reflection). For example, the code in **Figure 2** uses the nameof expression to extract the name of the parameter.

As the test method demonstrates, the ArgumentNullException's ParamName property has the value param1—a value set using the nameof(param1) expression in the method. The nameof expression isn't limited to parameters. You can use it to retrieve any programming element, as shown in **Figure 3**.

The nameof expression only retrieves the final identifier, even if you use more explicit dotted names. Also, in the case of attributes, the "Attribute" suffix isn't implied. Instead, it's required for compilation. It provides great opportunity to clean up messy code.

Primary Constructors

Auto-property initializers are especially useful in combination with primary constructors. Primary constructors give you a way to reduce ceremony on common object patterns. This feature has been significantly improved since May. Updates include:

1. An optional implementation body for the primary constructor: This allows for things such as primary constructor parameter validation and initialization, which was previously not supported.
2. Elimination of field parameters: declaration of fields via the primary constructor parameters. (Not moving forward with this feature as it was defined was the right decision, as it no longer forces particular naming conventions in ways that C# previously was ambivalent.)
3. Support for expression bodied functions and properties (discussed later in this article).

With the prevalence of Web services, multiple-tier applications, data services, Web API, JSON and similar technologies, one common form of class is the data transfer object (DTO). The DTO generally doesn't have much implementation behavior, but focuses on data storage simplicity. This focus on simplicity makes primary constructors compelling. Consider, for example, the immutable Pair data structure shown in this example:

```
struct Pair<T>(T first, T second)
{
  public T First { get; } = first;

  public T Second { get; } = second;

  // Equality operator ...
}
```

The constructor definition—Pair(string first, string second)—is merged into the class declaration. This specifies the constructor parameters are first and second (each of type T). Those parameters are also referenced in the property initializers and assigned to their corresponding properties. When you observe the simplicity of this class definition, its support for immutability and the requisite constructor (initializer for all properties/fields), you see how it helps you code correctly. That leads to a significant improvement in a common pattern that previously required unnecessary verbosity.

Primary constructor bodies specify behavior on the primary constructor. This helps you implement an equivalent capability on primary constructors as you can on constructors in general. For example, the next step in improving the reliability of the Pair<T> data structure might be to provide property validation. Such validation could ensure a value of null for Pair.First would be invalid. CTP3 now includes a primary constructor body—a constructor body without the declaration, as shown in **Figure 4**.

For clarity, I placed the primary constructor body at the first member of the class. However, this isn't a C# requirement. The primary constructor body can appear in any order relative to the other class members.

> Primary constructor bodies specify behavior on the primary constructor.

Although not functional in CTP3, another feature of read-only properties is that you can assign them directly from within the constructor (for example, First = first). This isn't limited to primary constructors, but is available for any constructor member.

An interesting consequence of support for auto-property initializers is that it eliminates many of the cases found in earlier versions where you needed explicit field declarations. The obvious case it doesn't eliminate is a scenario where validation on the setter is required. On the other hand, the need to declare read-only fields becomes virtually deprecated. Now, whenever a read-only field is declared, you can declare a read-only auto-property possibly as private, if that level of encapsulation is required.

The CompareTo method has parameters first and second—seemingly overlapping the parameter names of the primary constructor. Because primary constructor names are in scope within the auto-property initializers, first and second may seem ambiguous. Fortunately, this isn't the case. The scoping rules pivot on a different dimension you haven't seen in C# before.

Prior to C# 6.0, scope was always identified by the variable declaration placement within code. Parameters are bound within

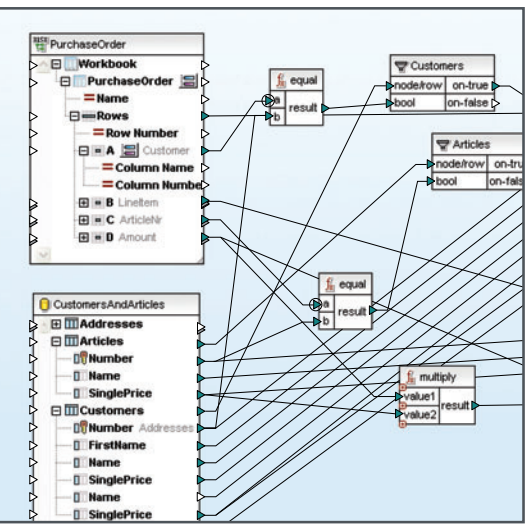
Figure 4 **Implementing a Primary Constructor Body**

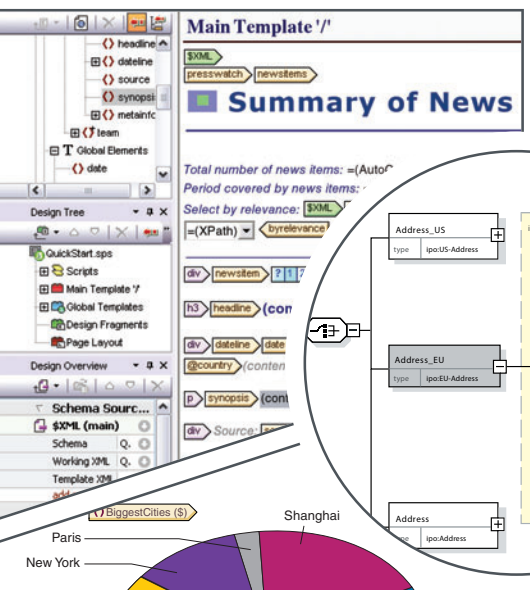
```
struct Pair<T>(T first, T second)
{
  {
    if (first == null) throw new ArgumentNullException("first");

    First = first; // NOTE: Not working in CTP3
  }

  public T First { get; }; // NOTE: Results in compile error for CTP3

  public T Second { get; } = second;

  public int CompareTo(T first, T second)
  {
    return first.CompareTo(First) + second.CompareTo(Second);
  }

// Equality operator ...
}
```

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

ALTOVA® MissionKit®

Experience how Altova MissionKit®, a software development suite of industrial-strength XML, SQL, and data integration tools, can simplify even the most advanced XML development projects.





Altova MissionKit includes multiple, tightly-integrated XML tools:

XMLSpy® – industry-leading XML editor

- Support for XML Schema 1.1 and XPath/XSLT/XQuery 3.0
- Industry's strongest validation engine with Smart Fix
- Powered by RaptorXML® for lightning-fast validation & processing
- Graphical editing views, powerful debuggers, code generation, & more

MapForce® – any-to-any data mapping & integration tool

- Drag-and-drop data conversion
- Mapping of XML, DBs, EDI, Excel®, XBRL, flat files & Web services
- Automation via MapForce Server

StyleVision® – visual XSLT stylesheet & report designer

- Graphical XSLT stylesheet & report design for XML, XBRL, & SQL databases
- XPath 3.0 and XSLT 3.0 support Powered by RaptorXML
- Output to HTML, PDF, Word & more
- Automation via StyleVision Server

Download a 30 day free trial!

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

the method they help declare, fields are bound within the class, and variables declared within an if statement are bound by the condition body of the if statement.

In contrast, primary constructor parameters are bound by time. The primary constructor parameters are only "alive" while the primary constructor is executing. This time frame is obvious in the case of the primary constructor body. It's perhaps less obvious for the auto-property initializer case.

> Primary constructor parameters are bound by time.

However, like field initializers translated to statements executing as part of a class initialization in C# 1.0+, auto-property initializers are implemented the same way. In other words, the scope of a primary constructor parameter is bound to the life of the class initializer and the primary constructor body. Any reference to the primary constructor parameters outside an auto-property initializer or primary constructor body will result in a compile error.

There are several additional concepts related to primary constructors that are important to remember. Only the primary constructor can invoke the base constructor. You do this using the base (contextual) keyword following the primary constructor declaration:

```
class UsbConnectionException(
  string message, Exception innerException, HidDeviceInfo hidDeviceInfo):
    Exception  (message, innerException)
{
  public HidDeviceInfo HidDeviceInfo { get;  } = hidDeviceInfo;
}
```

If you specify additional constructors, the constructor call chain must invoke the primary constructor last. This means a primary constructor can't have a this initializer. All other constructors must have them, assuming the primary constructor isn't also the default constructor:

```
public class Patent(string title, string yearOfPublication)
{
  public Patent(string title, string yearOfPublication,
    IEnumerable<string> inventors)
    ...this(title, yearOfPublication)
  {
    Inventors.AddRange(inventors);
  }
}
```

Hopefully, these examples help demonstrate that primary constructors bring simplicity to C#. They're an additional opportunity to do simple things simply, instead of simple things in a complex fashion. It's occasionally warranted for classes to have multiple constructors and call chains that make code hard to read. If you encounter a scenario where the primary constructor syntax makes your code look more complex rather than simplifying it, then don't use primary constructors. For all the enhancements to C# 6.0, if you don't like a feature or if it makes your code harder to read, just don't use it.

Expression Bodied Functions and Properties

Expression bodied functions are another syntax simplification in C# 6.0. These are functions with no statement body. Instead, you implement them with an expression following the function declaration.

For example, an override of ToString could be added to the Pair<T> class:

```
public override string ToString() => string.Format("{0}, {1}", First, Second);
```

There's nothing particularly radical about expression bodied functions. As with most of the features found in C# 6.0, they're intended to provide a simplified syntax for cases where the implementation is simple. The return type of the expression must, of course, match the return type identified in the function declaration. In this case, ToString returns a string, as does the function implementation expression. Methods that return void or Task should be implemented with expressions that don't return anything, either.

The expression bodied simplification isn't limited to functions. You can also implement read-only (getter only) properties using expressions—expression bodied properties. For example, you can add a Text member to the FingerPrint class:

```
public string Text =>
  string.Format("{0}: {1} - {2} ({3})", TimeStamp, Process, Config, User);
```

Other Features

There are several features no longer planned for C# 6.0:

- The indexed property operator ($) is no longer available and isn't expected for C# 6.0.
- The index member syntax isn't working in CTP3, although it's expected to return in a later release of C# 6.0:

```
var cppHelloWorldProgram = new Dictionary<int, string>
{
[10] = "main() {",
[20] = "    printf(\"hello, world\")",
[30] = "}"
};
```

- Field arguments in primary constructors are no longer part of C# 6.0.
- Both the binary numeric literal and the numeric separator ('_') within a numeric literal aren't currently certain to make it by release to manufacturing.

There are a number of features not discussed here because they were already covered in the May article, but static using statements (see itl.tc/?p=4038), declaration expressions (see itl.tc/?p=4040) and exception handling improvements (see itl.tc/?p=4042) are features that have remained stable.

Wrapping Up

Clearly, developers are passionate about C# and want to ensure it maintains its excellence. The language team is taking your feedback seriously and modifying the language as it processes what users have to say. Don't hesitate to visit roslyn.codeplex.com and let the team know your thoughts. Also, don't forget to check out itl.tc/csharp6 for updates on C# 6.0 until it's released. ■

Mark Michaelis *is the founder of IntelliTect. He also serves as the chief technical architect and trainer. Since 1996, he has been a Microsoft MVP for C#, Visual Studio Team System (VSTS), and the Windows SDK, and he was recognized as a Microsoft Regional Director in 2007. He also serves on several Microsoft software design-review teams, including C#, the Connected Systems Division, and VSTS. Michaelis speaks at developer conferences, has written numerous articles and books, and is currently working on the next edition of "Essential C#" (Addison-Wesley Professional).*

Thanks to the following Microsoft technical expert for reviewing this article: *Mads Torgersen*

BEST SELLER

ComponentOne Studio Enterprise 2014 v2 | from $1,315.60

ComponentOne a division of GrapeCity

.NET Tools for the Professional Developer: Windows, HTML5/Web, and XAML.

- Hundreds of UI controls for all .NET platforms including grids, charts, reports and schedulers
- Includes 40+ UI widgets built with HTML5, jQuery, CSS3 and SVG
- New Sparkline control for HTML5 and Web Forms and options for exporting Data views
- 40+ Windows 8.1 & Windows Phone 8.1 (beta) controls and Universal Windows app support
- All Microsoft platforms supported, Visual Studio 2013, ASP.NET, WinForms, WPF & more

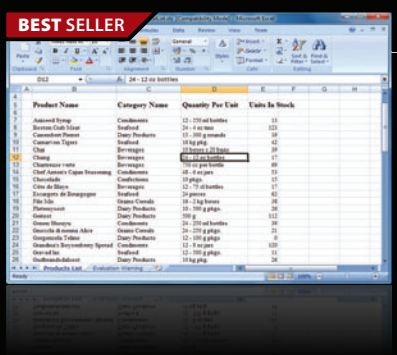
BEST SELLER

Help & Manual Professional | from $583.10

ec software

Easily create documentation for Windows, the Web and iPad.

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



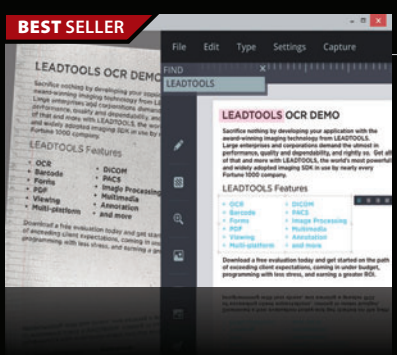
BEST SELLER

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

ASPOSE
Your File Format APIs

Every Aspose .NET component in one package.

- Programmatically manage popular file formats including Word, Excel, PowerPoint and PDF
- Work with charts, diagrams, images, project plans, emails, barcodes, OCR and OneNote files alongside many more document management features in .NET applications
- Common uses also include mail merging, adding barcodes to documents, building dynamic reports on the fly and extracting text from most document types



BEST SELLER

LEADTOOLS Document Imaging SDKs V18 | from $2,695.50

LEAD TECHNOLOGIES

Add powerful document imaging functionality to desktop, tablet, mobile & web applications.

- Comprehensive document image cleanup, preprocessing, viewer controls and annotations
- Fast and accurate OCR, OMR, ICR and Forms Recognition with multi-threading support
- PDF & PDF/A Read / Write / Extract / View / Edit
- Barcode Detect / Read / Write for UPC, EAN, Code 128, Data Matrix, QR Code, PDF417
- Zero-Footprint HTML5/JavaScript Controls & Native WinRT Libraries for Windows Store

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

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

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

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

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

GSA Schedule Contract GS-35F-0188R

Introduction to Async/Await on ASP.NET

Stephen Cleary

Most online resources around async/await assume you're developing client applications, but does async have a place on the server? The answer is most definitely "Yes." This article is a conceptual overview of asynchronous requests on ASP.NET, as well as a reference for the best online resources. I won't be covering the async or await syntax; I've already done that in an introductory blog post (bit.ly/19IkogW) and in an article on async best practices (msdn.microsoft.com/magazine/jj991977). This article focuses specifically on how async works on ASP.NET.

For client applications, such as Windows Store, Windows desktop and Windows Phone apps, the primary benefit of async is responsiveness. These types of apps use async chiefly to keep the UI responsive. For server applications, the primary benefit of async is scalability. The key to the scalability of Node.js is its inherently asynchronous nature; Open Web Interface for .NET (OWIN) was designed from the ground up to be asynchronous; and ASP.NET can also be asynchronous. Async: It's not just for UI apps!

This article discusses a preview release of ASP.NET vNext; information is subject to change.

This article discusses:

- Synchronous vs. asynchronous request handling on ASP.NET
- The advantages of asynchronous code for scaling
- Aborting requests
- Support for async and await

Technologies discussed:

ASP.NET 4.5, ASP.NET vNext

Synchronous vs. Asynchronous Request Handling

Before diving into asynchronous request handlers, I'd like to briefly review how synchronous request handlers work on ASP.NET. For this example, let's say the requests in the system depend on some external resource, like a database or Web API. When a request comes in, ASP.NET takes one of its thread pool threads and assigns it to that request. Because it's written synchronously, the request handler will call that external resource synchronously. This blocks the request thread until the call to the external resource returns. **Figure 1** illustrates a thread pool with two threads, one of which is blocked waiting for an external resource.

Eventually, that external resource call returns, and the request thread resumes processing that request. When the request is complete and the response is ready to be sent, the request thread is returned to the thread pool.

This is all well and good—until your ASP.NET server gets more requests than it has threads to handle. At this point, the extra requests have to wait for a thread to be available before they can run. **Figure 2** illustrates the same two-threaded server when it receives three requests.

In this situation, the first two requests are assigned threads from the thread pool. Each of these requests calls an external resource,

blocking their threads. The third request has to wait for an available thread before it can even start processing, but the request is already in the system. Its timer is going, and it's in danger of an HTTP Error 503 (Service unavailable).

But think about this for a second: That third request is waiting for a thread, when there are two other threads in the system effectively doing nothing. Those threads are just blocked waiting for an external call to return. They're not doing any real work; they're not in a running state and are not given any CPU time. Those threads are just being wasted while there's a request in need. This is the situation addressed by asynchronous requests.

Asynchronous request handlers operate differently. When a request comes in, ASP.NET takes one of its thread pool threads and assigns it to that request. This time the request handler will call that external resource asynchronously. This returns the request thread to the thread pool until the call to the external resource returns. **Figure 3** illustrates the thread pool with two threads while the request is asynchronously waiting for the external resource.

The important difference is that the request thread has been returned to the thread pool while the asynchronous call is in progress. While the thread is in the thread pool, it's no longer associated with that request. This time, when the external resource call returns, ASP.NET takes one of its thread pool threads and reassigns it to that request. That thread continues processing the request. When the request is completed, that thread is again returned to the thread pool. Note that with synchronous handlers, the same thread is used for the lifetime of the request; with asynchronous handlers, in contrast, different threads may be assigned to the same request (at different times).

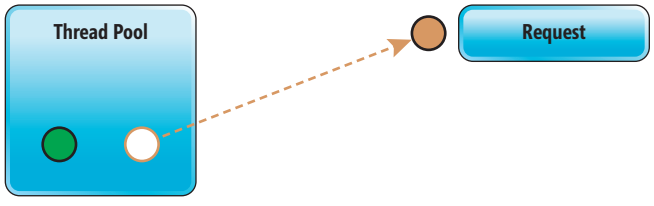


Figure 1 **Waiting Synchronously for an External Resource**

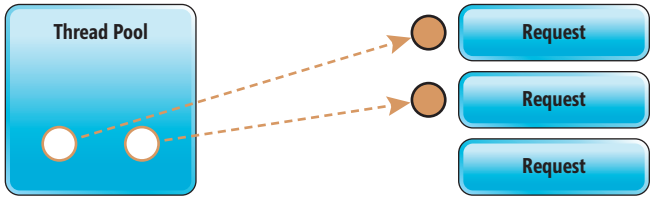


Figure 2 **A Two-Threaded Server Receiving Three Requests**

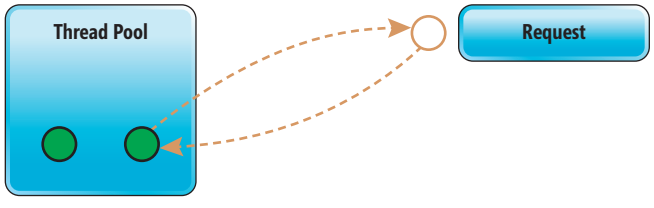


Figure 3 **Waiting Asynchronously for an External Resource**

Now, if three requests were to come in, the server could cope easily. Because the threads are released to the thread pool whenever the request has asynchronous work it's waiting for, they're free to handle new requests, as well as existing ones. Asynchronous requests allow a smaller number of threads to handle a larger number of requests. Hence, the primary benefit of asynchronous code on ASP.NET is scalability.

Why Not Increase the Thread Pool Size?

At this point, a question is always asked: Why not just increase the size of the thread pool? The answer is twofold: Asynchronous code scales both further and faster than blocking thread pool threads.

Asynchronous code can scale further than blocking threads because it uses much less memory; every thread pool thread on a modern OS has a 1MB stack, plus an unpageable kernel stack. That doesn't sound like a lot until you start getting a whole lot of threads on your server. In contrast, the memory overhead for an asynchronous operation is much smaller. So, a request with an asynchronous operation has much less memory pressure than a request with a blocked thread. Asynchronous code allows you to use more of your memory for other things (caching, for example).

Asynchronous code can scale faster than blocking threads because the thread pool has a limited injection rate. As of this writing, the rate is one thread every two seconds. This injection rate limit is a good thing; it avoids constant thread construction and destruction. However, consider what happens when a sudden flood of requests comes in. Synchronous code can easily get bogged down as the requests use up all available threads and the remaining requests have to wait for the thread pool to inject new threads. On the other hand, asynchronous code doesn't need a limit like this; it's "always on," so to speak. Asynchronous code is more responsive to sudden swings in request volume.

Bear in mind that asynchronous code does not replace the thread pool. This isn't thread pool *or* asynchronous code; it's thread pool *and* asynchronous code. Asynchronous code allows your application to make optimum use of the thread pool. It takes the existing thread pool and turns it up to 11.

What About the Thread Doing the Asynchronous Work?

I get asked this question all the time. The implication is that there must be some thread *somewhere* that's blocking on the I/O call to the external resource. So, asynchronous code frees up the request thread, but only at the expense of another thread elsewhere in the system, right? No, not at all.

To understand why asynchronous requests scale, I'll trace a (simplified) example of an asynchronous I/O call. Let's say a request needs to write to a file. The request thread calls the asynchronous write method. WriteAsync is implemented by the Base Class Library (BCL), and uses completion ports for its asynchronous I/O. So, the WriteAsync call is passed down to the OS as an asynchronous file write. The OS then communicates with the driver stack, passing along the data to write in an I/O request packet (IRP).

This is where things get interesting: If a device driver can't handle an IRP immediately, it must handle it asynchronously. So, the driver

tells the disk to start writing and returns a "pending" response to the OS. The OS passes that "pending" response to the BCL, and the BCL returns an incomplete task to the request-handling code. The request-handling code awaits the task, which returns an incomplete task from that method and so on. Finally, the request-handling code ends up returning an incomplete task to ASP.NET, and the request thread is freed to return to the thread pool.

Now, consider the current state of the system. There are various I/O structures that have been allocated (for example, the Task instances and the IRP), and they're all in a pending/incomplete state. However, there's no thread that is blocked waiting for that write operation to complete. Neither ASP.NET, nor the BCL, nor the OS, nor the device driver has a thread dedicated to the asynchronous work.

When the disk completes writing the data, it notifies its driver via an interrupt. The driver informs the OS that the IRP has completed, and the OS notifies the BCL via the completion port. A thread pool thread responds to that notification by completing the task that was returned from WriteAsync; this in turn resumes the asynchronous request code. There were a few threads "borrowed" for very short amounts of time during this completion-notification phase, but no thread was actually blocked while the write was in progress.

> Async and await on ASP.NET are all about I/O.

This example is drastically simplified, but it gets across the primary point: no thread is required for true asynchronous work. No CPU time is necessary to actually push the bytes out. There's also a secondary lesson to learn. Think about the device driver world, how a device driver must either handle an IRP immediately or asynchronously. Synchronous handling is not an option. At the device driver level, all non-trivial I/O is asynchronous. Many developers have a mental model that treats the "natural API" for I/O operations as synchronous, with the asynchronous API as a layer built on the natural, synchronous API. However, that's completely backward: in fact, the natural API is asynchronous; and it's the *synchronous* APIs that are implemented using *asynchronous* I/O!

Why Weren't There Asynchronous Handlers Already?

If asynchronous request handling is so wonderful, why wasn't it already available? Actually, asynchronous code is so good for scalability that the ASP.NET platform has supported asynchronous handlers and modules since the very beginnings of the Microsoft .NET Framework. Asynchronous Web pages were introduced in ASP.NET 2.0, and MVC got asynchronous controllers in ASP.NET MVC 2.

However, until recently, asynchronous code has always been awkward to write and difficult to maintain. Many companies decided it was easier all around to just develop the code synchronously and pay for larger server farms or more expensive hosting. Now, the tables have turned: in ASP.NET 4.5, asynchronous code using async and await is almost as easy as writing synchronous code. As large systems move into cloud hosting and demand more scale, more and more companies are embracing async and await on ASP.NET.

Asynchronous Code Is Not a Silver Bullet

As wonderful as asynchronous request handling is, it won't solve all your problems. There are a few common misunderstandings around what async and await can do on ASP.NET.

When some developers learn about async and await, they believe it's a way for the server code to "yield" to the client (for example, the browser). However, async and await on ASP.NET only "yield" to the ASP.NET runtime; the HTTP protocol remains unchanged, and you still have only one response per request. If you needed SignalR or AJAX or UpdatePanel before async/await, you'll still need SignalR or AJAX or UpdatePanel after async/await.

Asynchronous request handling with async and await can help your applications scale. However, this is scaling on a single server; you may still need to plan to scale *out*. If you do need a scale-out architecture, you'll still need to consider stateless, idempotent requests and reliable queueing. Async and await do help somewhat: they enable you to take full advantage of your server resources, so you won't have to scale out as often. But if you do need to scale out, you'll need a proper distributed architecture.

Async and await on ASP.NET are all about I/O. They really excel at reading and writing files, database records, and REST APIs. However, they're not good for CPU-bound tasks. You *can* kick off some background work by awaiting Task.Run, but there's no point in doing so. In fact, that will actually hurt your scalability by interfering with the ASP.NET thread pool heuristics. If you have CPU-bound work to do on ASP.NET, your best bet is to just execute it directly on the request thread. As a general rule, don't queue work to the thread pool on ASP.NET.

Finally, consider the scalability of your system as a whole. A decade ago, a common architecture was to have one ASP.NET Web server that talked to one SQL Server database back end. In that kind of simple architecture, usually the database server is the scalability bottleneck, not the Web server. Making your database calls asynchronous would probably not help; you could certainly use them to scale the Web server, but the database server will prevent the system as a whole from scaling.

Rick Anderson makes the case *against* asynchronous database calls in his excellent blog post, "Should My Database Calls Be Asynchronous?" (bit.ly/1rw66UB). There are two arguments that support this: first, asynchronous code is difficult (and therefore expensive in developer time compared to just purchasing larger servers); and second, scaling the Web server makes little sense if the database back end is the bottleneck. Both of those arguments made perfect sense when that post was written, but both arguments have weakened over time. First, asynchronous code is much easier to write with async and await. Second, the data back ends for Web sites are scaling as the world moves to cloud computing. Modern back ends such as Microsoft Azure SQL Database, NoSQL and other APIs can scale much further than a single SQL Server, pushing the bottleneck back to the Web server. In this scenario, async/await can bring a tremendous benefit by scaling ASP.NET.

wijmo 5

A New Generation of JavaScript Controls

A collection of UI controls for mobile and web application development.

Touch First, Mobile First

True JavaScript Controls

First Class Angular Support

Optimized for Size & Speed

FlexGrid for JavaScript

Download a free trial at
wijmo.com

wijmo
from ComponentOne
a division of GrapeCity

Before Getting Started

The first thing you need to know is that async and await are only supported on ASP.NET 4.5. There's a NuGet package called Microsoft.Bcl.Async that enables async and await for the .NET Framework 4, but do not use it; it will not work correctly! The reason is that ASP.NET itself had to change the way it manages its asynchronous request handling to work better with async and await; the NuGet package contains all the types the compiler needs but will *not* patch the ASP.NET runtime. There is no workaround; you need ASP.NET 4.5 or higher.

Next, be aware that ASP.NET 4.5 introduces a "quirks mode" on the server. If you create a new ASP.NET 4.5 project, you don't have to worry. However, if you upgrade an existing project to ASP.NET 4.5, the quirks are all turned on. I recommend you turn them all off by editing your web.config and setting httpRuntime.targetFramework to 4.5. If your application fails with this setting (and you don't want to take the time to fix it), you can at least get async/await working by adding an appSetting key of aspnet:UseTaskFriendlySynchronizationContext with value "true." The appSetting key is unnecessary if you have httpRuntime.targetFramework set to 4.5. The Web development team has a blog post on the details of this new "quirks mode" at bit.ly/1pbmnzK. Tip: If you're seeing odd behavior or exceptions, and your call stack includes LegacyAspNetSynchronizationContext, your application is running in this quirk mode. LegacyAspNetSynchronizationContext isn't compatible with async; you need the regular AspNetSynchronizationContext on ASP.NET 4.5.

In ASP.NET 4.5, all the ASP.NET settings have good default values for asynchronous requests, but there are a couple of other settings you might want to change. The first is an IIS setting: consider raising the IIS/HTTP.sys queue limit (Application Pools | Advanced Settings | Queue Length) from its default of 1,000 to 5,000. The other is a .NET runtime setting: ServicePointManager.DefaultConnectionLimit, which has a default value of 12 times the number of cores. The DefaultConnectionLimit limits the number of simultaneous outgoing connections to the same hostname.

A Word on Aborting Requests

When ASP.NET processes a request synchronously, it has a very simple mechanism for aborting a request (for example, if the request exceeded its timeout): It will abort the worker thread for that request. This makes sense in the synchronous world, where each request has the same worker thread from beginning to end. Aborting threads isn't wonderful for long-term stability of the AppDomain, so by default ASP.NET will regularly recycle your application to keep things clean.

With asynchronous requests, ASP.NET won't abort worker threads if it wants to abort a request. Instead, it will cancel the request using a CancellationToken. Asynchronous request handlers should accept and honor cancellation tokens. Most newer frameworks (including Web API, MVC and SignalR) will construct and pass you a CancellationToken directly; all you have to do is declare it as a parameter. You can also access ASP.NET tokens directly; for example, HttpRequest.TimedOutToken is a CancellationToken that cancels when the request times out.

As applications move into the cloud, aborting requests becomes more important. Cloud-based applications are more dependent on external services that may take arbitrary amounts of time. For example, one standard pattern is to retry external requests with exponential backoff; if your application depends on multiple services like this, it's a good idea to apply a timeout cap for your request processing as a whole.

Current State of Async Support

Many libraries have been updated for compatibility with async. Async support was added to Entity Framework (in the EntityFramework NuGet package) in version 6. You do have to be careful to avoid lazy loading when working asynchronously, though, because lazy loading is always performed synchronously. HttpClient (in the Microsoft.Net.Http NuGet package) is a modern HTTP client designed with async in mind, ideal for calling external REST APIs; it's a modern replacement for HttpWebRequest and WebClient. The Microsoft Azure Storage Client Library (in the WindowsAzure.Storage NuGet package) added async support in version 2.1.

Newer frameworks such as Web API and SignalR have full support for async and await. Web API in particular has built its entire pipeline around async support: not only async controllers, but async filters and handlers, too. Web API and SignalR have a very natural async story: you can "just do it" and it "just works."

This brings us to a sadder story: Today, ASP.NET MVC only partially supports async and await. The basic support is there—async controller actions and cancellation work appropriately. The ASP.NET Web site has an absolutely excellent tutorial on how to use async controller actions in ASP.NET MVC (bit.ly/1m1LXTx); it's the best resource for getting started with async on MVC. Unfortunately, ASP.NET MVC does not (currently) support async filters (bit.ly/1oAyHLc) or async child actions (bit.ly/1px47RG).

ASP.NET Web Forms is an older framework, but it also has adequate support for async and await. Again, the best resource for getting started is the tutorial on the ASP.NET Web site for async Web Forms (bit.ly/Ydho7W). With Web Forms, async support is opt-in. You have to first set Page.Async to true, then you can use PageAsyncTask to register async work with that page (alternatively, you can use async void event handlers). PageAsyncTask also supports cancellation.

If you have a custom HTTP handler or HTTP module, ASP.NET now supports asynchronous versions of those, as well. HTTP handlers are supported via HttpTaskAsyncHandler (bit.ly/1nWpWFj) and HTTP modules are supported via EventHandlerTaskAsyncHelper (bit.ly/1m1Sn4O).

As of this writing, the ASP.NET team is working on a new project known as ASP.NET vNext. In vNext, the entire pipeline is asynchronous by default. Currently, the plan is to combine MVC and Web API into a single framework that has full support for async/await (including async filters and async view components). Other async-ready frameworks such as SignalR will find a natural home in vNext. Truly, the future is async.

Respect the Safety Nets

ASP.NET 4.5 introduced a couple of new "safety nets" that help you catch asynchronous problems in your application. These are on by default, and should stay on.

GdPicture.NET 10

- Document viewing, processing, printing and scanning (TWAIN & WIA).
- Reading, writing and converting vector and raster images in more than 90 formats, PDF included.
- OMR, OCR, barcode reading and writing (linear & 2D).
- Annotations for image and PDF within Windows and Web applications.
- Color detection engine for image and PDF compression.

And much more ...

All-In-One Document Imaging SDK

Royalty-Free Document Imaging Toolkits for .NET and COM/ActiveX

GdPicture.NET 10 Plugins

- NEW Color detection
- NEW DICOM image reader
- NEW MICR reader
- Full managed PDF support
- Full annotations support for PDF and images
- OCR
- Forms processing
- JBIG2 encoding
- 1D and 2D barcode reading and writing

Try GdPicture.NET 10 FREE for 30 days

www.gdpicture.com

When a synchronous handler attempts to perform asynchronous work, you'll get an InvalidOperationException with the message, "An asynchronous operation cannot be started at this time." There are two primary causes for this exception. The first is when a Web Forms page has async event handlers, but neglected to set Page.Async to true. The second is when the synchronous code calls an async void method. This is yet another reason to avoid async void.

The other safety net is for asynchronous handlers: When an asynchronous handler completes the request, but ASP.NET detects asynchronous work that hasn't completed, you get an InvalidOperationException with the message, "An asynchronous module or handler completed while an asynchronous operation was still pending." This is usually due to asynchronous code calling an async void method, but it can also be caused by improper use of an Event-based Asynchronous Pattern (EAP) component (bit.ly/19VdUWu).

There's an option you can use to turn off both safety nets: HttpContext.AllowAsyncDuringSyncStages (it can also be set in web.config). A few pages on the Internet suggest setting this whenever you see these exceptions. I can't disagree more vehemently. Seriously, I don't know why this is even possible. Disabling the safety nets is a horrible idea. The only possible reason I can think of is if your code is already doing some extremely advanced asynchronous stuff (beyond anything I've ever attempted), and you are a multithreading genius. So, if you've read this entire article yawning and thinking, "Please, I'm no n00b," then I suppose you can consider disabling the safety nets. For the rest of us, this is an extremely dangerous option and should not be set unless you're fully aware of the ramifications.

Getting Started

Finally! Ready to get started taking advantage of async and await? I appreciate your patience.

First, review the "Asynchronous Code Is Not a Silver Bullet" section in this article to ensure async/await is beneficial to your architecture. Next, update your application to ASP.NET 4.5 and turn off quirks mode (it's not a bad idea to run it at this point just to make sure nothing breaks). At this point, you're ready to start true async/await work.

Start at the "leaves." Think about how your requests are processed and identify any I/O-based operations, especially anything network-based. Common examples are database queries and commands and calls to other Web services and APIs. Choose one to start with, and do a bit of research to find the best option for performing that operation using async/await. Many of the built-in BCL types are now async-ready in the .NET Framework 4.5; for example, SmtpClient has the SendMailAsync methods. Some types have async-ready replacements available; for example, HttpWebRequest and WebClient can be replaced with HttpClient. Upgrade your library versions, if necessary; for example, Entity Framework got async-compatible methods in EF6.

However, avoid "fake asynchrony" in libraries. Fake asynchrony is when a component has an async-ready API, but it's implemented by just wrapping the synchronous API within a thread pool thread. That is counterproductive to scalability on ASP.NET. One prominent example of fake asynchrony is Newtonsoft JSON.NET, an otherwise excellent library. It's best to not call the (fake) asynchronous versions for serializing JSON; just call the synchronous versions instead. A trickier example of fake asynchrony is the BCL file streams. When a file stream is opened, it must be *explicitly* opened for asynchronous access; otherwise, it will use fake asynchrony, synchronously blocking a thread pool thread on the file reads and writes.

Once you've chosen a "leaf," then start with a method in your code that calls into that API, and make it into an async method that calls the async-ready API via await. If the API you're calling supports CancellationToken, your method should take a CancellationToken and pass it along to the API method.

Whenever you mark a method async, you should change its return type: void becomes Task, and a non-void type T becomes Task<T>. You'll find that then all the callers of that method need to become async so they can await the task, and so on. Also, append Async to the name of your method, to follow the Task-based Asynchronous Pattern conventions (bit.ly/1uBKGKR).

Allow the async/await pattern to grow up your call stack toward the "trunk." At the trunk, your code will interface with the ASP.NET framework (MVC, Web Forms, Web API). Read the appropriate tutorial in the "Current State of Async Support" section earlier in this article to integrate your async code with your framework.

Along the way, identify any thread-local state. Because asynchronous requests may change threads, thread-local state such as ThreadStaticAttribute, ThreadLocal<T>, thread data slots and CallContext.GetData/SetData will not work. Replace these with HttpContext.Items, if possible; or you can store immutable data in CallContext.LogicalGetData/LogicalSetData.

Here's a tip I've found useful: you can (temporarily) duplicate your code to create a vertical partition. With this technique, you don't change your synchronous methods to asynchronous; you copy the entire synchronous method and then change the copy to be asynchronous. You can then keep most of your application using the synchronous methods and just create a small vertical slice of asynchrony. This is great if you want to explore async as a proof-of-concept or do load testing on just part of the application to get a feeling for how your system could scale. You can have one request (or page) that's fully asynchronous while the rest of your application remains synchronous. Of course, you don't want to keep duplicates for every one of your methods; eventually, all the I/O-bound code will be async and the synchronous copies can be removed.

Wrapping Up

I hope this article has helped you get a conceptual grounding in asynchronous requests on ASP.NET. Using async and await, it's easier than ever to write Web applications, services and APIs that make maximum use of their server resources. Async is awesome! ■

Stephen Cleary *is a husband, father and programmer living in northern Michigan. He has worked with multithreading and asynchronous programming for 16 years and has used async support in the Microsoft .NET Framework since the first community technology preview. His homepage, including his blog, is at stephencleary.com.*

Thanks to the following Microsoft technical expert for reviewing this article: *James McCaffrey*

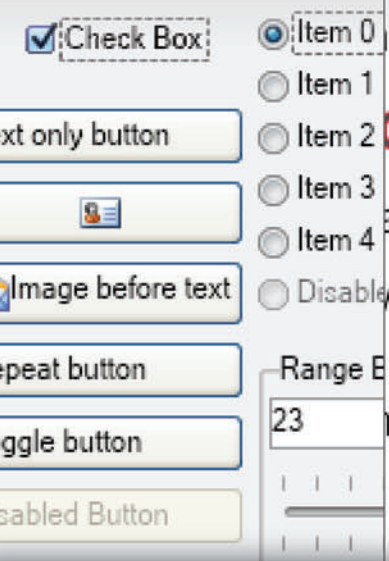
the myth: “To create a .NET application that runs on WinForms, WPF, Silverlight, MonoMac, Xamarin.Mac you have to write it 5 times (once for each platform)”

the truth: Don’t trust old-school devs. With Nevron Open Vision you only write once!

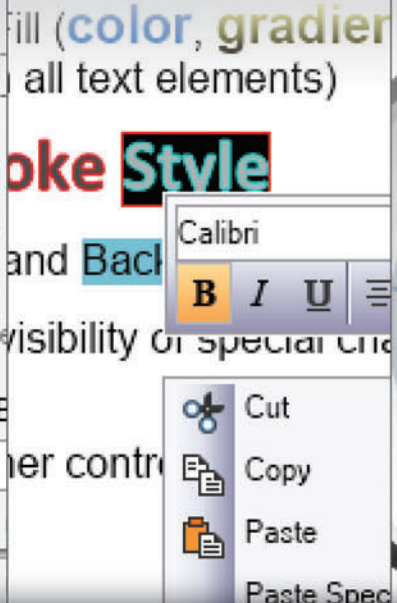
NEVRON OPEN VISION for .NET

comes with:

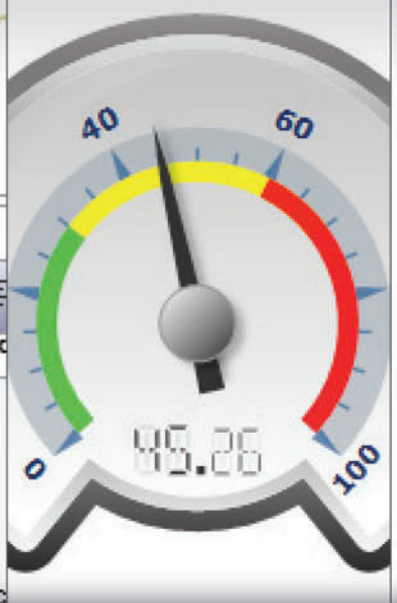
UI SUITE



TEXT EDITOR



GAUGE



BARCODE

COMING SOON: NOV CHART for .NET NOV DIAGRAM for .NET

Learn more at **www.nevron.com** today

www.nevron.com | email@nevron.com | +1 888-201-6088 (Toll free, USA and Canada)

Connect Your IoT Device to the Cloud

Steven Edouard and Bruno Terkaly

We recently introduced the idea of connecting an Internet of Things (IoT) device—in our case the Raspberry Pi—to the cloud (Microsoft Azure) to deliver a push notification to your mobile device whenever somebody rings your doorbell at home. This lets you see who's at your front door step at home from anywhere in the world using your mobile device.

In the September 2014 article, "Soup to Nuts: From Raw Hardware to Cloud-Enabled Device" (msdn.microsoft.com/magazine/dn781356), we walked through the process of integrating the device to storage blobs using Azure Mobile Services to obtain a Shared Access Signature (SAS). This let the device directly upload the file to a storage container without any service tier. Now we can upload the photo to the cloud, but there are still a few things left to do to send a notification to the device.

So far, Azure Mobile Services and Azure Storage have been leveraged in this project. You'll also need to take advantage of the service bus queues and scheduled tasks services to send messages from the Raspberry Pi device to your mobile device. These two services will be integrated with Azure Mobile Services, which will receive messages from the service bus queue and record them in a database. In the spirit of open source and new-age solutions, we'll use a MongoDB database hosted and maintained by MongoLab that you can add for free as an Azure Add-on.

This article discusses:

- Using service bus queueing to handle message traffic
- Writing policies to secure message transfer
- Storing and managing messages for easy retrieval

Technologies discussed:

Microsoft Azure, Azure Mobile Services, MongoDB, Node.js

Service Bus for Distributed Messaging

In the context of IoT computing, the service bus queueing capabilities provide powerful abstractions. Service bus queueing supports asynchronous messaging between endpoints and can scale to support virtually any workload. It also lets applications communicate between the cloud and on-premises without requiring virtual networking, which can be a logistical and security nightmare.

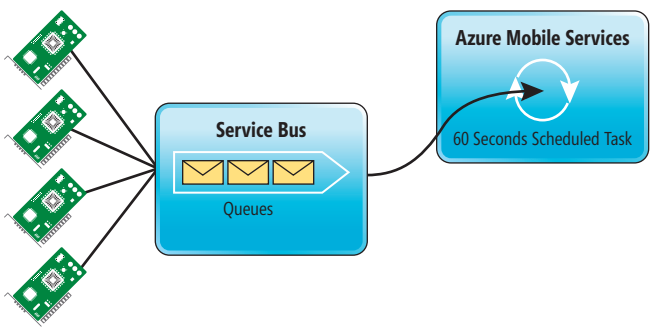


Figure 1 **Architectural Diagram of Azure Mobile Services Reading from Service Bus**

Figure 2 **Azure Mobile Scheduled Task doorbellListener**

```
// Get a reference to the azure module
var azure = require('azure');
// Get our service bus connection string
var connectionString = process.env.ServiceBusConnectionString;
// This task will run for 60 seconds so the initial timeout should be 60
var c_Timeout = 60;
function doorbellListener() {
  //Get the current unix time in seconds
  var date = new Date();
  var time = date.getTime();
  var startSeconds = time / 1000;

  var sb = azure.createServiceBusService(connectionString);
  listenForMessages(c_Timeout);

  // Define a function that will listen for messages
  // on the queue for number of seconds
  function listenForMessages(seconds) {
    console.log('Doorbell Listener Started for timeout: ' + seconds);
    // Long poll the service bus for seconds
    sb.receiveQueueMessage("smartdoor", { timeoutIntervalInS: seconds },
      function(err, data) {
        if(err){
          // This path will get hit if we didn't get any messages
          console.log(err);
        }
        else{
          // We have received a message from a device
          var ringNotification = JSON.parse(data.body);
          console.log('recieved message from doorbell ' +
            ringNotification.doorbellId + '
            with image link ' + ringNotification.imagePointer)
          function continueListeningForMessages(){

            // Go back and listen for more messages for the duration of this task
            var currentDate = new Date();
            var currentSeconds = currentDate.getTime() / 1000;

          console.log('currentSeconds ' + currentSeconds);
          console.log('startSeconds ' + startSeconds);

        // Compute the seconds between when we started this scheduled task and now
        // This is the time that we will long-poll the service bus
        var newTimeout = Math.round((c_Timeout - (currentSeconds - startSeconds)));
          if(newTimeout > 0){
          // Note: the receiveQueueMessage function takes ints no decimals!!
          listenForMessages(newTimeout);
        }
      }
```

We addressed some of these issues in the February 2014 column, "The Windows Azure Service Bus and the Internet of Things" (msdn.microsoft.com/magazine/dn574801). Azure offers another queueing service called storage queues, which has similar capabilities, but differs in a few key ways. We chose to go with service bus queues because of publisher/subscriber capabilities, greater scalability in message handling and the ability to easily convert to other service bus services. You can read more about storage queues at bit.ly/XSJB43.

Service bus queues expose a regular RESTful API, which supports the ability to receive messages by long polling. This is a technique of opening an HTTP connection for a certain period of time. Long polling is a great technique for IoT computing scenarios because it supports timeouts. This lets devices close connections until the next long poll, providing relief to power consumption and network resources.

We'll have a many-to-one relationship of SmartDoor doorbell devices to a single mobile service. More specifically, this form of pub/sub will involve many message publishers and only a single message subscriber. The subscriber in this case will be the mobile service, which will read from the queue. One or more SmartDoor devices will be the publisher. It's entirely possible to do the reverse, in case you'd like to send a message back to the device.

There are four types of communication patterns typically used in IoT scenarios. As you can see in **Figure 1**, one or more Raspberry Pi devices are publishing messages to the service bus queues. Azure Mobile Services is the only subscriber to service bus queues.

Azure Mobile Services provides an integrated task scheduler, which can schedule tasks at fixed intervals. We'll read from the service bus queue at an interval of 60 seconds. Check our blog post at bit.ly/1o1nUrY for more detailed information on connecting mobile services to a service bus queue. With our scheduled task, doorBellListener (see **Figure 2**), we'll read from the queue by using the service bus API in the Node.js Azure SDK with our connection string.

Because of the asynchronous nature of Node.js and the behavior of sending a message to the service bus via long polling, you must compute the timeout value that's passed to recieveQueueMessage, depending on how long this instance of the scheduled task has been running. This will prevent multiple instances of the task from running simultaneously.

The nice thing about service bus is it exposes a RESTful API. No matter what device you're using, you can send it messages. Azure has SDKs for major languages such as Python, Ruby, Node.js and C#. Because we want the code to be translated to any platform, we'll use the RESTful API directly.

In order to send a message, we'll need to create a policy for the service bus queue. This policy is a key that permits certain operations on your service bus. There are a certain number of policies for any given queue. Take that into account in your own problem domain.

We'll generate a policy called DevicePolicy that only lets users send messages to the service bus queue (see **Figure 3**).

This ensures that if the key ever gets into the wrong hands, nobody else can listen to messages on the service bus. The device code for the Raspberry Pi to send a message the service bus queue is shown in **Figure 4**.

You can see the full code of program.cs at bit.ly/1qFYAF2. In that code, we do a POST request on the service bus queue RESTful API and provide an SAS, similar to the signature we received from mobile services for blob storage. This SAS is composed of a cryptographic key using the SHA-256 algorithm. It defines the resource being accessed as well as the SAS expiration time. The createToken method is a simple method that constructs the SAS based on your Shared Access Key from the DevicePolicy policy.

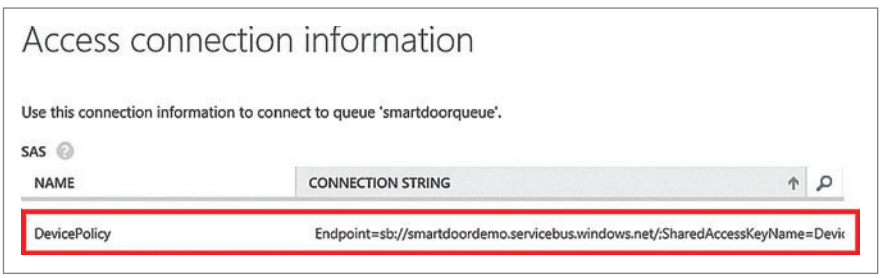


Figure 3 **A DevicePolicy Will Send Messages to the Service Bus Queue**

Figure 4 **Sending a Message Indicating Successful Photo Upload**

```
Console.WriteLine("Sending notification to service bus queue");
  WebRequest sbRequest = WebRequest.Create(
    "https://smartdoordemo.servicebus.Windows.net/smartdoorqueue/messages");
  var headers = sbRequest.Headers;
  sbRequest.Method = "POST";
  using (var sbMessageStream = sbRequest.GetRequestStream())
  {
    string body = JsonConvert.SerializeObject(new DoorBellNotification()
    {
      doorBellID = deviceID,
      imageUrl = photoResp.photoId
    });
    var bytes = Encoding.UTF8.GetBytes(body);
    sbMessageStream.Write(bytes, 0, bytes.Length);
    headers.Add("Authorization", createToken(
      "https://smartdoordemo.servicebus.Windows.net/smartdoorqueue/
      messages", "DevicePolicy",
      ConfigurationManager.AppSettings["ServiceBusSharedAccessKey"]));
  }

  try
  {
    Console.WriteLine("Sending door bell notification for " + deviceID);
    using (var response = sbRequest.GetResponse())
    {
      Console.WriteLine("Sucessfully Sent");
    }
  }
  catch (Exception e)
  {
    Console.WriteLine("Couldn't post to service bus -" + e);
  }
```

After constructing the SAS, we place it in an HTTP header and place the serialized message (in JSON format) in the request body. After the POST request is made, a message is sent to the service bus queue to indicate the successful upload of a photo. The message contains the link to the uploaded blob and a unique identifier for this doorbell device which comes from the application settings. This identifier can be found in the app.config xml file, which sits next to the doorbell listener executable:

```
<appSettings>
  <add key="DoorBellID" value="123456"/>
...
</appSettings>
```

Now the device is sending messages to the service bus when it runs. By going to the Log tab in our Mobile Services landing page we can access the live service log output. When we run the C# code in Visual Studio you can see the mobile service got the message contents through the log output in the portal.

Although Azure Mobile Services offers perfectly good tables (backed by SQL Server) for storage, we're going for a slightly different approach. We'll use MongoDB as our database solution.

Figure 5 **Schemas Defined in Mongoose**

```
var photoSchema = mongoose.Schema({
                url : String,
                timestamp: String
            });
var doorbellSchema = mongoose.Schema({
            doorBellID: String,
            photos: [photoSchema]
        });

var Photo = mongoose.model('Photo', photoSchema)
var DoorBell = mongoose.model('DoorBell', doorbellSchema);

// Expose these schemas
exports.DoorBell = DoorBell;
exports.Photo = Photo;
```

MongoDB works very well with Node.js because it's a document-oriented database and resembles a collection of JSON objects in storage. Read more about the MongoDB open source project at mongodb.org. We'll use MongoLab, a Database as a Service (DaaS) provider, to host our database.

We need the database to keep track of a couple things:

- DoorBells—An individual Raspberry Pi device
- Pictures—An individual picture taken by a doorbell

Once the MongoDB database is set up, we can use Mongoose as our MongoDB Driver for our Azure Mobile Services Node.js code by installing it to our mobile services Git repository. This is the same process we went through to install the qs node module:

```
npm install mongoose
```

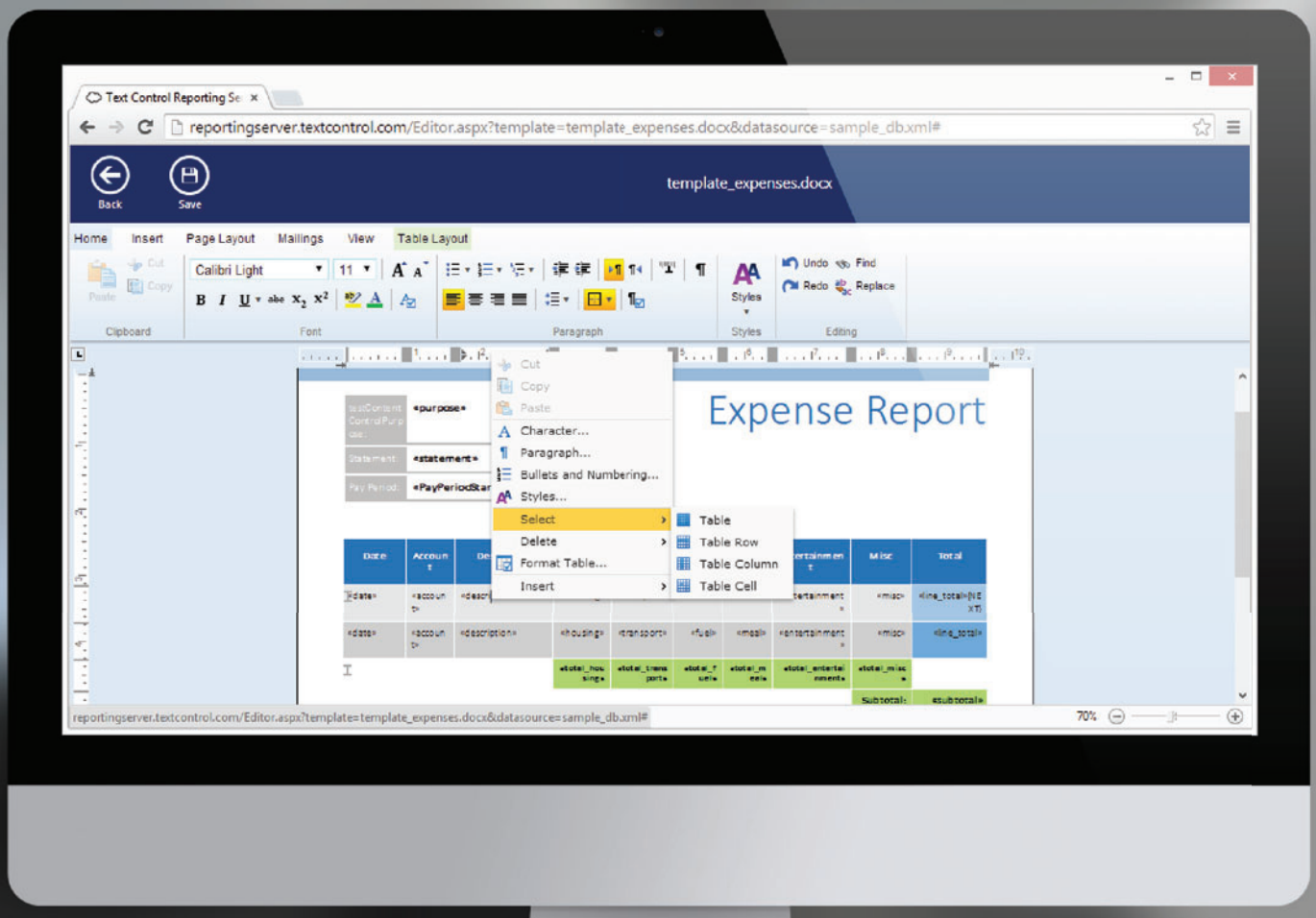
Pushing the local repository will trigger the mobile service to install Mongoose to its repository. This is how we can get any Node.js package into Azure Mobile Services. As with every Node.js module, we can reference it using RequireJs and initialize Mongoose with our MongoDB connection string in our scheduled task:

```
var mongoose = require('mongoose');
```

Mongoose will be responsible for creating a structured schema in our document database (see **Figure 5**). We'll use two entities: doorbell and photo. Every doorbell object in our database will represent one Raspberry Pi device. It will contain a unique identifier,

Figure 6 **Logic to Verify Doorbell and Manage Photo**

```
// Create database entry for this image
dbConnectAndExecute(function(err){

  if(err){
    console.log('could not record image to database -' + err);
    return;
  }
  DoorBell.findOne({ doorBellID: ringNotification.doorBellID},
    function(err, doorbell){
    if(err){
      return console.error(err);
    }

    if(doorbell === null){
      console.log('Doorbell not found in DB, creating a new one');
      // Take the entire body's json, assuming it fits into this schema
      var entityObject = {
        doorBellID : ringNotification.doorBellID,
        photos : []
      };
      entityObject.photos.push({
        url : ringNotification.imageUrl,
        // Set timestamp to current server time
        timestamp : ((new Date()).getTime()).toString()
      })
      var doorbell = new DoorBell(entityObject);
    }
    else{
      // We already have this device in the database—add a picture
      doorbell.photos.push({
        url : ringNotification.imageUrl,
        // Set timestamp to current server time
        timestamp : ((new Date()).getTime()).toString()
      });
    }
    // Commit changes to db
    doorbell.save(function (err, entity) {
    if(err){
      return console.error(err);
    }
    console.log('sucessfully created new entity for: ' + entity);
    return;
  });
  });
});
```



Cross-browser, cross-platform document and template editing

HTML5-BASED TX TEXT CONTROL

The first true WYSIWYG, HTML5-based Web editor and reporting template designer for ASP.NET.

Give your users an MS Word compatible editor to create powerful reporting templates anywhere - in any browser on any device.

Download your 30-day trial version today:
www.textcontrol.com/html5

Visual Studio Partner

the new TEXT CONTROL

```
input from pin gpio17, value was False
output to pin gpio4, value was True
input from pin gpio22, value was True
output to pin gpio4, value was False
Pushing photo to SAS Url: https://smartdoor.blob.core.windows.net:443/democontai
ner/49826.jpg?st=2014-08-06T05%3A23%3A06Z&se=2014-08-06T05%3A38%3A06Z&sp=w&sr=b&
su=2012-02-12&sig=A3bOCr7XOp4171FAuXU46N0qwfr7tMSkD1E9hW4hahU%3D
Writing photo to blob...
Sucessfully Uploaded Photo to cloud
Sucessfully sent service bus message
```

Figure 7 **The Outcome of Sending a Photo and Message**

doorBellID, and an array of photo objects. Each photo contains a link to the photo in blob storage, as well as a timestamp generated by the service when it receives a service bus message.

We expose the DoorBell and Photo schemas publicly via the exports keyword. Notice how photoSchema is interleaved within doorbellSchema. This is how the data will be reflected when stored in the database.

We'll place this code in the shared folder of our mobile services repository. This lets us use the schemas anywhere in our service. To reference the schemas in our scheduled task, we just need to import it with a require statement:

```
var schemas = require('../shared/schemas.js');
```

Now we can use these schemas in our scheduled task to define new database entities. We'll use a specialized function to ensure we're connected to MongoDB and execute a callback. Upon receiving a message from the service bus, the mobile service should:

1. Check if the doorbell with the specified doorBellID in the message exists in the database.
2. If it doesn't exist, create a new doorbell entity with the photo.
3. If it does exist, just append the new photo to the existing doorbell's photo array.

You can see the logic for this in the code in **Figure 6**.

You can see a full demo of doorbelllistener at bit.ly/VKrlYU.

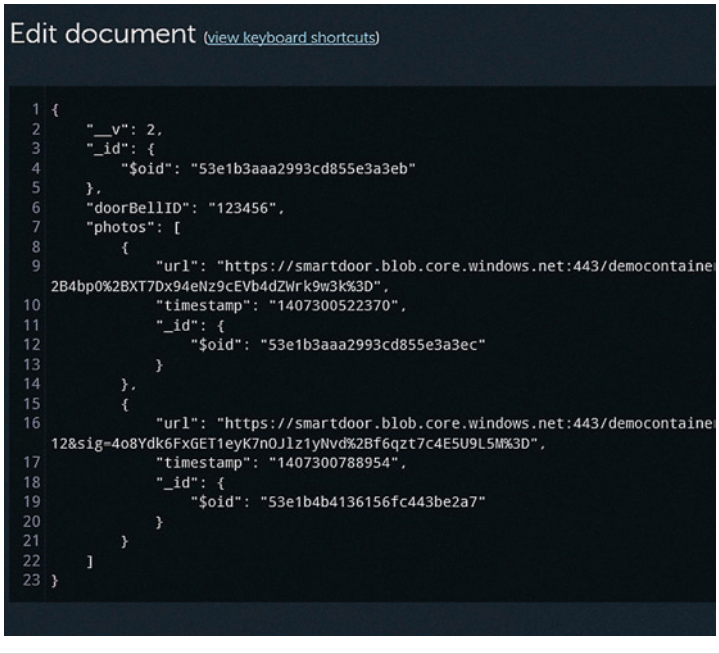


Figure 8 **MongoDB Registering Entries**

We call the dbConnectAndExecute function to ensure we're connected to our MongoDB database in MongoLabs. Then we query the database for a doorbell with the ID specified in the message. If the query comes up empty, we create a new doorbell entity. Otherwise, we append the photo to the fetched entity and commit the changes to the database.

Now see in **Figure 7** what happens when we send a photo and service bus message with our Raspberry Pi device.

Inspecting the mobile services log in the portal after Raspberry Pi has sent the message shows the photo was processed and added to the database.

For good measure, we can check our MongoDB database to ensure we're actually tracking the photos. MongoLab provides a rich MongoDB inspection interface via its Web portal. **Figure 8** shows what the doorbell entity may look like after a couple photo uploads.

Now, Raspberry Pi is fully integrated to our cloud service. The device can upload a file directly to cloud storage, notify the cloud service via service bus and have the information stored in a database.

The next step in this series will be integrating a mobile application to the cloud back end with push notifications of the images. We'll use custom APIs to surface objects from the database to the app and integrate a third-party API for image facial recognition and a notification hub to power the push notifications. You can examine the code repositories to build and tweak the solution yourself. The Azure Mobile Services back end is at bit.ly/1mHClOq. You'll find the Raspberry Pi Code at bit.ly/1l8aOtF.

Integrating Raspberry PI devices to cloud back ends is critical. Leveraging open source technologies such as Node.js makes sense in many scenarios where you're using a lightweight back end to scale to a large number of devices.

The Azure service bus provides a safe and convenient way for occasionally connected devices to leverage a reliable messaging infrastructure that can scale to many clients. Finally, leveraging NoSQL stores is a popular way to persist data and retain native JavaScript syntax in both the data layer and back-end Node.js service. ■

Steven Edouard *is a developer evangelist at Microsoft. Before that, he worked on the .NET runtime team as a software test engineer delivering products like the .NET Framework 4.5 and .NET Native Compilation. Now his passion resides in exciting people on the limitless potentials of cloud computing services through technical demonstrations, online content and presentations.*

Bruno Terkaly *is a developer evangelist for Microsoft. His depth of knowledge comes from years of experience in the field, writing code using a multitude of platforms, languages, frameworks, SDKs, libraries and APIs. He spends time writing code, blogging and giving live presentations on building cloud-based applications, specifically using the Microsoft Azure platform. You can read his blog at blogs.msdn.com/b/brunoterkaly.*

Thanks to the following Microsoft technical experts for reviewing this article: *Gil Isaacs and Brent Stineman*

Work With Relational Data, Not Complex APIs or Services

Whether you are a developer using ADO.NET, JDBC, OData, or MySQL, or a systems integrator working with SQL Server or Biztalk, or even an information worker familiar with ODBC or Excel – our products give you bi-directional access to live data through easy-to-use technologies that you are already familiar with. If you can connect to a database, then you will already know how to connect to Salesforce, SAP, SharePoint, Dynamics CRM, Google Apps, QuickBooks, and much more!

Give RSSBus a try today and see what mean:

visit us online at www.rssbus.com to learn more or download a free trial.

rssbus

INTEGRATION YOUR WAY

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

ORLANDO
ROYAL PACIFIC RESORT AT UNIVERSAL ORLANDO

NOV 17-21

"HELP US VISUAL STUDIO LIVE! – YOU'RE OUR ONLY HOPE!"

THE CODE IS STRONG WITH THIS ONE!

While your development challenges may not be as dire as the Rebellion's, we know you're searching for real-world knowledge and education on the Microsoft Platform. And for 21 years and counting, Visual Studio Live! has been the conference more developers trust for independent, practical training.

EVENT PARTERS Magenic Microsoft

PLATINUM SPONSOR esri

GOLD SPONSORS New Relic. sitecore

SUPPORTED BY msdn magazine Visual Studio Visual Studio MAGAZINE

LIVE! 360

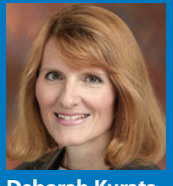
TECH EVENTS WITH PERSPECTIVE

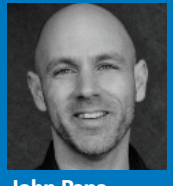
FEATURED SPEAKERS:


Andrew Brust


Rockford Lhotka


Miguel Castro


Deborah Kurata


John Papa


Rachel Appel


Brian Randell


Leonard Lobel


Brian Noyes

REGISTER TODAY AND SAVE $300!

Use promo code ORLOCT4 by October 15

5 GREAT CONFERENCES. 1 GREAT PRICE.

Visual Studio Live! Orlando is part of Live! 360, the ultimate IT and Developer line-up. This means you'll have access to 5 co-located conferences for 1 low price!

Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS

SQL Server LIVE!
TRAINING FOR DBAs AND IT PROS

SharePoint LIVE!
TRAINING FOR COLLABORATION

Modern Apps LIVE!
MOBILE, CROSS-DEVICE & CLOUD DEVELOPMENT

TECHMENTOR
IN-DEPTH TRAINING FOR IT PROS

TURN THE PAGE FOR MORE EVENT DETAILS

PRODUCED BY 1105 MEDIA

vslive.com/orlando | live360events.com

AGENDAS AT-A-GLANCE Visual Studio Live! & Modern Apps Live!

Visual Studio / .NET Framework	JavaScript / HTML5 Client	ASP.NET	Cross-Platform Mobile Development	Cloud Computing	Windows Client	Windows Phone	Data Management

Modern Apps LIVE!

START TIME	END TIME	Visual Studio Live! & Modern Apps Live! Pre-Conference: Sunday, November 16, 2014				
4:00 PM	9:00 PM	Pre-Conference Registration - Royal Pacific Resort Conference Center				
6:00 PM	9:00 PM	Dine-A-Round Dinner @ Universal CityWalk				

START TIME	END TIME	Visual Studio Live! & Modern Apps Live! Pre-Conference Workshops: Monday, November 17, 2014 (Separate entry fee required)				
6:30 AM	8:00 AM	Pre-Conference Workshop Registration - Coffee and Morning Pastries				
8:00 AM	5:00 PM	**VSM01** - Workshop: Native Mobile App Development for iOS, Android, and Windows using C# - *Marcel de Vries & Rockford Lhotka*		**VSM02** - Workshop: AngularJS in 0 to 60 - *John Papa*	**VSM03** - Workshop: From Code to Release: DevOps for Developers - *Brian Randell*	**MAM01** - Workshop: Modern App Technology Overview - Android, iOS, Cloud, and Mobile Web - *Nick Landry, Kevin Ford, & Steve Hughes*
5:00 PM	6:00 PM	EXPO Preview				
6:00 PM	7:00 PM	Live! 360 Keynote: To Be Announced				

START TIME	END TIME	Visual Studio Live! & Modern Apps Live! Day 1: Tuesday, November 18, 2014				
7:00 AM	8:00 AM	Registration - Coffee and Morning Pastries				
8:00 AM	9:00 AM	Visual Studio Live! & Modern Apps Live! Keynote: To Be Announced				
9:00 AM	9:30 AM	Networking Break • Visit the EXPO				
9:30 AM	10:45 AM	**VST01** - Getting Started with Xamarin - *Walt Ritscher*	**VST02** - Great User Experiences with CSS 3 - *Robert Boedigheimer*	**VST03** - What's New in MVC 5 - *Miguel Castro*	**VST04** - New IDE and Editor Features in Visual Studio 2013 - *Deborah Kurata*	**MAT01** - Defining Modern App Development - *Rockford Lhotka*
11:00 AM	12:15 PM	**VST05** - Building Your First Universal Application for Windows Phone and Windows Store - *Brian Peek*	**VST06** - HTML5 and Modernizr for Better Web Sites - *Robert Boedigheimer*	**VST07** - What's New in Web API 2 - *Miguel Castro*	**VST08** - Visual Studio Data Tools for Developers - *Deborah Kurata*	**MAT02** - Modern App Architecture - *Rockford Lhotka*
12:15 PM	2:00 PM	Lunch • Visit the EXPO				
2:00 PM	3:15 PM	**VST09** - Introduction to Developing Mobile Apps for the Web Developer - *Ben Hoelting*	**VST10** - Build an Angular and Bootstrap Web Application in Visual Studio from the Ground Up - *Deborah Kurata*	**VST11** - ASP.NET MVC for Mobile Devices - *Rob Daigneau*	**VST12** - The Road to Continuous Delivery, Automated UI Testing for Web, WPF and Windows Forms - *Marcel de Vries*	**MAT03** - ALM with Visual Studio Online (TFS) and Git - *Brian Randell*
3:15 PM	4:15 PM	Networking Break • Visit the EXPO				
4:15 PM	5:30 PM	**VST13** - Creating Responsive Cross-Platform Native/Web Apps with JavaScript and Bootstrap - *Ben Dewey*	**VST14** - Hate JavaScript? Try TypeScript - *Ben Hoelting*	**VST15** - What's New in WPF 4.5 - *Walt Ritscher*	**VST16** - Understanding Dependency Injection & Writing Testable Software - *Miguel Castro*	**MAT04** - Reusing Business and Data Access Logic Across Platforms - *Kevin Ford*
5:30 PM	7:30 PM	Exhibitor Reception				

START TIME	END TIME	Visual Studio Live! & Modern Apps Live! Day 2: Wednesday, November 19, 2014				
7:00 AM	8:00 AM	Registration - Coffee and Morning Pastries				
8:00 AM	9:00 AM	Live! 360 Keynote: To Be Announced				
9:15 AM	10:30 AM	**VSW01** - Build Cross-Platform Apps with Shared Projects, CSLA .NET, and C# - *Rockford Lhotka*	**VSW02** - AngularJS Jump-Start - *John Papa*	**VSW03** - Best Practices for Self Hosting Web API Based Services - *Rob Daigneau*	**VSW04** - The Busy Developers Guide to Virtualization with Hyper-V - *Brian Randell*	**MAW01** - Coding for Quality and Maintainability - *Jason Bock*
10:30 AM	11:00 AM	Networking Break • Visit the EXPO				
11:00 AM	12:15 PM	**VSW05** - Building Mobile Cross-Platform Apps with HTML5 & Cordova in Visual Studio - *Nick Landry*	**VSW06** - AngularJS Anywhere with Node.js - *John Papa*	**VSW07** - Slice Development Time With ASP.NET MVC, Visual Studio, and Razor - *Philip Japikse*	**VSW08** - Build It and Ship It with TFS and Release Management - *Brian Randell*	**MAW02** - UX Design for Modern Apps - *Anthony Handley*
12:15 PM	1:45 PM	Birds-of-a-Feather Lunch • Visit the EXPO				
1:45 PM	3:00 PM	**VSW09** - What's New in Azure for Developers - *Vishwas Lele*	**VSW10** - I Just Met You, and "This" is Crazy, But Here's My NaN, So Call(me) Maybe? - *Rachel Appel*	**VSW11** - Automated Cross Browser Testing of Your Web Applications with Visual Studio CodedUI - *Marcel de Vries*	**VSW12** - Readable Code - *John Papa*	**MAW03** - Applied UX: iOS, Android, Windows - *Anthony Handley*
3:00 PM	4:00 PM	Networking Break • Visit the EXPO • Expo Raffle @ 3:30 p.m.				
4:00 PM	5:15 PM	**VSW13** - API Management - *Vishwas Lele*	**VSW14** - Creating HTML5 and Angular Websites Using Visual Studio LightSwitch - *Michael Washington*	**VSW15** - Build Real-Time Websites and Apps with SignalR - *Rachel Appel*	**VSW16** - Visual Studio Online: An Update Every Three Weeks - *Brian Randell*	**MAW04** - Leveraging Azure Services - *Kevin Ford*
8:00 PM	10:00 PM	Live! 360 Evening Event				

START TIME	END TIME	Visual Studio Live! & Modern Apps Live! Day 3: Thursday, November 20, 2014				
7:30 AM	8:00 AM	Registration - Coffee and Morning Pastries				
8:00 AM	9:15 AM	**VSH01** - Creating Visual Studio Cloud Business Applications for Office 365 / SharePoint 2013 - *Michael Washington*	**VSH02** - To Be Announced	**VSH03** - Games Development with Unity and Other Frameworks for Windows and Windows Phone - *Brian Peek*	**VSH04** - Managing the .NET Compiler - *Jason Bock*	**MAH01** - Analyzing Results with Power BI - *Steve Hughes*
9:30 AM	10:45 AM	**VSH05** - Microsoft Azure Web Sites for the Web Developer - *Eric D. Boyd*	**VSH06** - Building Rich Data Input Windows 8 Applications - *Brian Noyes*	**VSH07** - Build Your First Mobile App in 1 Hour with Microsoft App Studio - *Nick Landry*	**VSH08** - Writing Asynchronous Code Using .NET 4.5 and C# 5.0 - *Brian Peek*	**MAH02** - Building a Native iOS App
11:00 AM	12:15 PM	**VSH09** - Moving Web Apps to the Cloud - *Eric D. Boyd*	**VSH10** - XAML Antipatterns - *Ben Dewey*	**VSH11** - To Be Announced	**VSH12** - Asynchronous Debugging in .NET - *Jason Bock*	**MAH03** - Building an Android App with Xamarin - *Nick Landry*
12:15 PM	1:30 PM	Lunch on the Lanai				
1:30 PM	2:45 PM	**VSH13** - Node.js for .NET Developers - *Jason Bock*	**VSH14** - Building Maintainable and Extensible MVVM WPF Apps with Prism 5 - *Brian Noyes*	**VSH15** - Learning Entity Framework 6 - *Leonard Lobel*	**VSH16** - Rock Your Code Using Code Contracts - *David McCarter*	**MAH04** - Building a Windows App - *Brent Edwards*
3:00 PM	4:15 PM	**VSH17** - Building Mobile Cross-Platform Apps in C# with Azure Mobile Services - Nick Landy	**VSH18** - XAML for the WinForms Developer - *Philip Japikse*	**VSH19** - Database Development with SQL Server Data Tools - *Leonard Lobel*	**VSH20** - Rock Your Code With Visual Studio Add-ins - *David McCarter*	**MAH05** - Building a Responsive Single Page App - *Allen Conway*
4:30 PM	5:45 PM	Live! 360 Conference Wrap-up				

START TIME	END TIME	Visual Studio Live! & Modern Apps Live! Post-Conference Workshops: Friday, November 21, 2014 (Separate entry fee required)				
7:30 AM	8:00 AM	Post-Conference Workshop Registration - Coffee and Morning Pastries				
8:00 AM	5:00 PM	**VSF01** - Workshop: Service Orientation Technologies: Designing, Developing, & Implementing WCF and the Web API - *Miguel Castro*		**VSF02** - Workshop: Build a Windows 8.1 Application in a Day - *Philip Japikse*		**MAF01** - Workshop: Modern App Development In-Depth: iOS, Android, Windows, and Web - *Brent Edwards, Anthony Handley, & Allen Conway*

Speakers and sessions subject to change

Developing Your First Game with Unity and C#, Part 3

Adam Tuliper

You're still with me in this series. Good. In the first article, I covered some Unity basics (msdn.microsoft.com/magazine/dn759441). In the second, I focused on 2D in Unity (msdn.microsoft.com/magazine/dn781360). Now I get to my favorite part of game development—3D. The world of 3D is a truly magical place—amazing immersive environments, rich sound effects and beautiful visuals—even just a simple puzzle game with real-world physics can keep you hooked for hours.

3D games definitely add a layer of complexity over 2D, but by taking it piece by piece you can build up a cool 3D game. New project settings for both 2D and 3D in Unity support 3D. You can have 3D objects in a 2D game (and vice versa).

What Makes Up a 3D Scene?

3D scenes consist primarily of three main visual components—lights, mesh renderers and shaders. A light is, well, a light, and Unity supports four different types. You can find them all under the GameObject menu. Experiment with adding the various types and changing their properties. The easiest one to light up your scene is a directional light, which is like the sun in the sky.

A mesh (or model) is a collection of vertices that make up the polygons that make up an object. A shader is a compiled routine that contains code to control how your object will show or interact with light. Some shaders simply take light and reflect it like a mirror; others take a texture (an image to be applied to your mesh) and can enable shadows and depth; and some even allow you to cut visual holes through your models, like a fence.

Models are typically FBX or OBJ files exported from another modelling software package. FBX files can also contain animation data, so you might receive one FBX file for your model and one containing several animations. Several third-party file formats are also supported, such as the Autodesk Maya .ma format and Blender files. You will typically need the third-party program installed on the same system if you want Unity to import these files, and then it's simply a matter of dragging and dropping them into your Unity project, just as you would any other file. Behind the scenes, Unity will convert other file formats (upon import or detecting file changes) into the FBX file format.

Asset Store

I touched on the Asset Store in my first article, but in 3D games is where it's really handy. I'm not an artist, and because this is a technical magazine, I assume most of you aren't, either. (If you are,

This article discusses:

- Rotating and moving objects
- Animations and terrains
- The Unity coordinate system
- Threading and coroutines

Technologies discussed:

Unity, C#, Microsoft .NET Framework

please accept my congrats, you are part of a rare group.) But if I want to create a game with lush environments and old destroyed buildings, for example, it's not a problem. I can buy what I need from the Asset Store. If I want 15 different zombies, I can procure a pack from Mixamo in the Asset Store. The potential combinations are nearly endless, so don't worry about someone else's game looking like yours. Best of all, the Asset Store integrates into Unity. You can upgrade your packages by clicking Window | Asset Store and then the bin icon. You can also check out reviews and comments to more easily determine if a particular item is good for your project, for example, whether its mobile-optimized or not. Desktop games can typically handle a lot more objects/vertices/textures/memory than a mobile game, although some of the newer chips make mobile devices today seem like Xbox 360s.

In a typical 3D game, many of the same concepts from a 2D game apply—colliders, triggers, rigid bodies, game objects/transforms, components and more. Regardless of the type of 3D game, you'll typically want to control input, movement, and characters; use animations and particle effects; and build an imaginative world that's both fantastical and realistic. I'll discuss some of the ways Unity helps with this.

Figure 1 **Various Methods for Moving Objects**

```
// Method 1
void Update()
{
  // Move from point a to point b by .2 each frame - assuming called in Update.
  // Will not overshoot the destination, so .2 is the max amount moved.
  transform.position =
    Vector3.MoveTowards(transform.position, new Vector3(10, 1, 100), .2f);

}
// Method 2
void Update()
{
  // Interpolate from point a to point b by a percentage each frame,
  // in this case 10 percent (.1 float).
  var targetPosition = new Vector3(10,0,15);
  transform.position = Vector3.Lerp(parentRig.position, targetPosition, .1f);
}

// Method 3
void Update()
{
  // Teleport the object forward in the direction it is rotated.
  // If you rotate the object 90 degrees, it will now move forward in the direction
  // it is now facing. This essentially translates local coordinates to
  // world coordinates to move object in direction and distance
  // specified by vector. See the Unity Coordinate Systems section in the
  // main article.
  transform.Translate(Vector3.forward * Time.deltaTime);
}

// Method 4
void FixedUpdate()
{
  // Cause the object to act like it's being pushed to the
  // right (positive x axis). You can also use (Vector.right * someForce)
  // instead of new Vector().
  rigidbody.AddForce( new Vector3(7, 0, 0), ForceMode.Force);
}

// Method 5
void FixedUpdate()
{
  // Cause the object to act like it's being pushed to the positive
  // x axis (world coordinates) at a speed of approx 7 meters per second.
  // The object will slow down due to friction.
  rigidbody.velocity = new Vector3(7,0,0);
}

// Method 6
// Move the rigidbody's position (note this is not via the transform).
// This method will push other objects out of the way and move to the right in
// world space ~three units per second.
private Vector3 speed = new Vector3(3, 0, 0);
void FixedUpdate()
{
  rigidbody.MovePosition(rigidbody.position + speed * Time.deltaTime);
}

// Method 7
void FixedUpdate()
{
  // Vector3.forward is 0,0,1. You could move a character toward 0,0,1, but you
  // actually want to move the object forward no matter its rotation.
  // This is used when you want a character to move in the direction it's
  // facing, no matter its rotation. You need to convert the meaning of
  // this vector from local space (0,0,1) to world space,
  // and for that you can use TransformDirection and assign that vector
  // to its velocity.
  rigidbody.velocity = transform.TransformDirection(Vector3.forward * speed);
}
```

Input, Movement and Character Controllers

Reading input for movement becomes a bit more complicated in 3D because rather than simply moving in the X and Y planes, you can now move in three dimensions: X, Y and Z. Scenarios for 3D movement include (but aren't limited to) top-down movement, where a character moves only horizontally and vertically; rotating a camera or character when reading mouse input, as is done in many first-person shooter (FPS) games; strafing left to right when reading horizontal input; rotating to turn around when reading horizontal input; or just walking backward. There are a good number of movement options from which to choose.

When moving an object, you don't give it a position to move to, as you might expect. Remember, you're executing code with each frame, so you need to move the object in small increments. You can either let the physics engine handle this by adding a force to your rigidbody to move it, or you can tween the object. Tweening basically means transitioning between values; that is, moving from point A to point B. There are various ways to tween values in Unity, including free third-party libraries such as iTween. **Figure 1** shows some manual ways to move an object in Unity. Note that for simplicity, they haven't been optimized (to do so, I'd hold a reference to the transform in a variable to prevent going from managed code to native code too often).

Each approach has advantages and disadvantages. There can be a performance hit moving just the transform (methods 1-2), though it's a very easy way to do movement. Unity assumes if an object doesn't have a rigidbody component on it, it probably isn't a moving object. It builds a static collision matrix internally to know where objects are, which enhances performance. When you move objects by moving the transform, this matrix has to be recalculated, which causes a performance hit. For simple games, you may never notice the hit and it may be the easiest thing for you to do, although as your games get more complicated, it's important to move the rigidbody itself, as I did in methods 4-6.

Rotating Objects

Rotating an object is fairly simple, much like moving an object, except the vectors now represent degrees instead of a position or a normalized vector. A normalized vector is simply a vector with

Figure 2 **Methods for Rotating an Object**

```
// Any code below that uses _player assumes you
// have this code prior to it to cache a reference to it.
private GameObject _player;
void Start()
{
  _player = GameObject.FindGameObjectWithTag("Player");
}


// Method 1
void Update () {
  // Every frame rotate around the X axis by 1 degree a
  // second (Vector3.right = (1,0,0)).
  transform.Rotate(Vector3.right * Time.deltaTime);
}

// Method 2
void Update () {
  // No matter where the player goes, rotate toward him, like a gun
  // turret following a target.
  transform.LookAt(_player.transform);
}

// Method 3
void Update()
{
  Vector3 relativePos = _player.transform.position - transform.position;
  // If you set rotation directly, you need to do it via a Quaternion.
  transform.rotation = Quaternion.LookRotation(relativePos);
}
```

a max value of one for any value and can be used when you just want to simply reference a direction by using a vector. There are some vector keywords available to help, such as Vector3.right, back, forward, down, up, left, right, zero and one. Anything that will move or rotate in the positive horizontal direction can use Vector.right, which is just a shortcut for (1,0,0), or one unit to the right. For rotating an object, this would represent one degree. In **Figure 2**, I just rotate an object by a little bit in each frame.

Each of these techniques has minor nuances. Which one should you use? I would try to apply forces to the rigidbody, if possible. I've probably just confused you a bit with that option. The good news is, there's existing code that can do virtually all of this for you.

Did you notice the Quaternion in Method 3? Unity uses Quaternions internally to represent all rotations. Quaternions are efficient structures that prevent an effect called gimbal lock, which can happen if you use regular Euler angles for rotation. Gimbal lock occurs when two axes are rotated to be on the same plane and then can't be separated. (The video at bit.ly/1mKgdFl provides a good explanation.) To avoid this problem, Unity uses Quaternions rather than Euler angles, although you can specify Euler angles in the Unity Editor and it will do the conversion into a Quaternion on the back end. Many people never experience gimbal lock, but I wanted to point out that if you want to set a rotation directly in code, you must do it via a Quaternion, and you can convert from Euler angles using Quaternion.Euler.

Now that you've seen many options, I should note that I find the easiest method is to use a rigidbody and simply apply .AddForce to the character. I prefer to reuse code when I can, and luckily Unity supplies a number of prefabs.

Let's Not Reinvent the Wheel

Unity provides the Sample Assets package in the Asset Store (bit.ly/1twXOKr), which contains a cross-platform input manager with mobile joystick controls, some animations and particles, and most important, some prebuilt character controllers.

There are some older assets included with Unity (as of this writing, version 4.6). Those assets are now distributed as a separate package that Unity can update separately. Rather than having to write all of the code to create a first-person character in your game, a third-person character, or even a self-driving car, you can simply use the prefabs from the sample assets. Drag and drop into your scene and instantly you have a third person view with multiple animations and full access to the source code, as shown in **Figure 3**.

Animations

An entire book could be dedicated (and has) to the Mecanim animation system in Unity. Animations in 3D are generally more complicated than in 2D. In 2D, an animation file typically changes a sprite renderer in each key frame to give the appearance of animation. In 3D, the animation data is a lot more complex. Recall from my second article that animation files contain key frames. In 3D, there can be many key frames, each with many data points for changing a finger, moving an arm or a leg, or for performing any number and type of movements. Meshes can also have defined bones in them and can use components called skinned mesh renderers, which deform the mesh based on how the bones move, much as a living creature would.

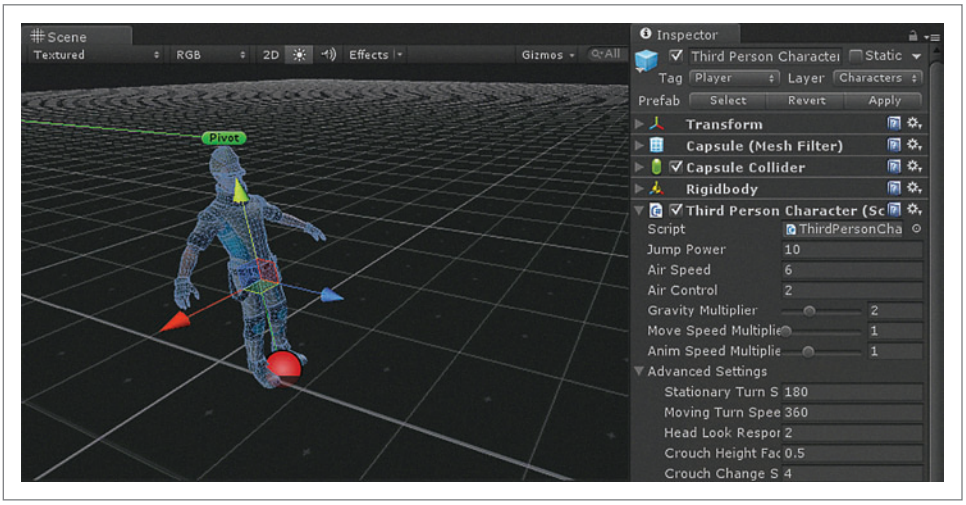


Figure 3 **A Third-Person Prefab**

Animation files are usually created in a third-party modeling/animation system, although you can create them in Unity, as well.

The basic pose for a character in a 3D animation system is the T-pose, which is just what it sounds

Spreadsheets Made Easy.

Fastest Calculations

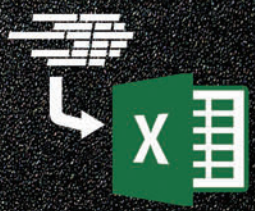
Evaluate complex Excel-based models and business rules with the fastest and most complete Excel-compatible calculation engine available.

Powerful Controls

Add powerful Excel-compatible viewing, editing, formatting, calculating, filtering, sorting, charting, printing and more to your WinForms, WPF and Silverlight applications.

Comprehensive Charting

Enable users to visualize data with comprehensive Excel-compatible charting which makes creating, modifying, rendering and interacting with complex charts easier than ever before.



Scalable Reporting

Easily create richly formatted Excel reports without Excel from any ASP.NET, Windows Forms, WPF or Silverlight application.

Download your free fully functional evaluation at SpreadsheetGear.com

SpreadsheetGear

Toll Free USA (888) 774-3273 | Phone (913) 390-4797 | sales@spreadsheetgear.com

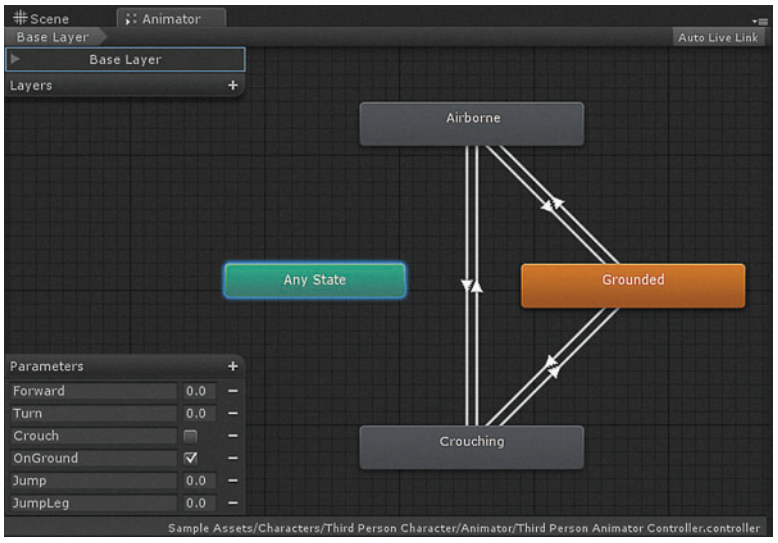


Figure 4 **Animation Controller for Controlling a Character's Animation States**

like—the character standing straight with outstretched arms, and it applies to just about any humanoid-shape model. You can then enliven that basic character by having Mecanim assign virtually any animation file to it. You can have a zombie, elf and human all dancing the same way. You can mix and match the animation files however you see fit and assign them via states much as you would in 2D. To do this, you use an animation controller like the one shown in **Figure 4**.

Remember, you can get characters and animations from the Unity Asset Store; you can create them with modeling tools; and there are third-party products like Mixamo's Fuse that enable you to quickly generate your own customized characters. Check out my Channel 9 videos for an intro to animation in Unity.

Creating a World

Unity has a built-in terrain system for generating a world. You can create a terrain and then use the included terrain tools to sculpt your terrain, make mountains, place trees and grass, paint textures, and more. You can add a sky to your world by importing the skybox package (Assets | Import Package | Skyboxes) and assigning it in Edit | Render Settings | Skybox Material. It took me just a couple of minutes to create a terrain with reflective, dynamic water, trees, sand, mountains and grass, as shown in **Figure 5**.

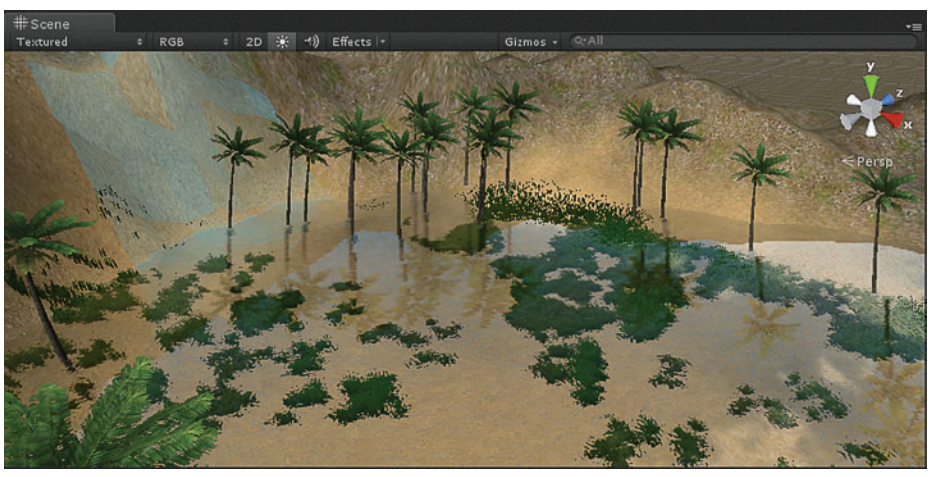


Figure 5 **A Quickly Created Terrain**

Unity Coordinate Systems

Unity has four different methods for referring to a point in a game or on the screen as shown in **Figure 6**. There's screen space, which ranges from 0 to the number of pixels and is used typically to get the location on the screen where the user touches or clicks. The viewport space is simply a value from 0 to 1, which makes it easy to say, for example, that halfway is .5, rather than having to divide pixels by 2. So I can easily place an object in the middle of the screen by using (.5, .5) as its position. World space refers to the absolute positioning of an object in a game based on three coordinates, (0, 0, 0). All top-level game objects in a scene have their coordinates listed in world space. Finally, local space is always relative to the parent game object. With a top-level game object, this is the same as world space. All child game objects are listed in the Editor in coordinates relative to their parent, so a model in your app of a house, for example, may have world coordinates of (200, 0, 35), while its front door (assuming it's a child game object of the house) might be only (1.5, 0, 0), as that's relative to the parent. In code, when you reference transform.position, it's always in world coordinates, even if it's a child object. In the example, the door would be (201.5, 0, 35), but if you instead reference transform.localPosition, you'd return (1.5, 0, 0). Unity has functions for converting among the various coordinate systems.

In the prior move examples I mostly moved using world space, but in some cases used local space. Refer back to method 7 in **Figure 1**. In that example I take a local normalized (or unit) vector of Vector.forward, which is (0,0,1). This by itself doesn't have much meaning. However, it shows intent to move something on the Z axis, which is forward. What if the object is rotated 90 degrees from (0,0,0)? Forward can now have two meanings. It can mean the original absolute Z axis (in world coordinates), or a Z axis relative to the rotated object, which is always pointing forward for the object. If I want an object to always move forward no matter its rotation, I can simply translate between local forward to the real-world forward

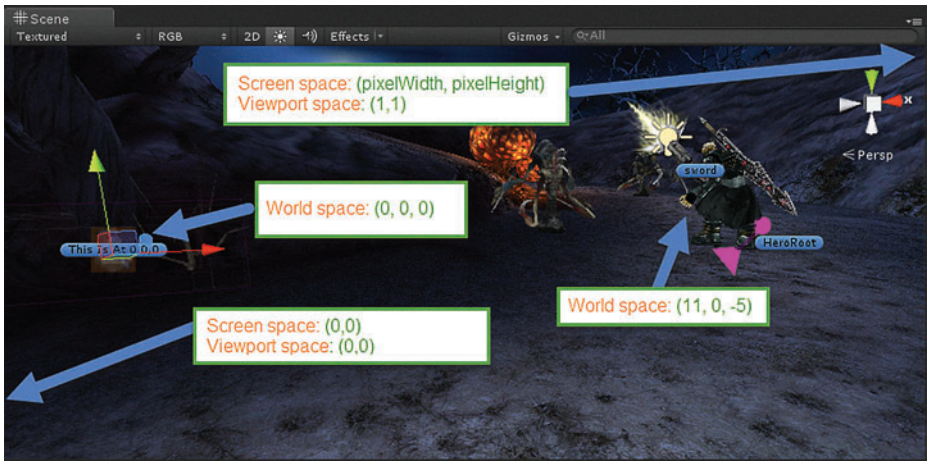


Figure 6 **Coordinates in Unity**

vector by using transform.TransformDirection(Vector3.forward * speed) as is shown in that example.

Threading and Coroutines

Unity uses a coroutine system to manage its threads. If you want something to happen in what you think should be a different thread, you kick off a coroutine rather than creating a new thread. Unity manages it all behind the scenes. What happens is the coroutine pauses when it hits the yield method. In the example in **Figure 7**, an attack animation is played, paused for a random length and then played in attack again.

Figure 7 **Using a Coroutine to Pause Action**

```
void Start()
{
  // Kick off a separate routine that acts like a separate thread.
  StartCoroutine(Attack());
}

IEnumerator Attack()
{
  // Trigger an attack animation.
  _animator.SetTrigger("Attack");
  // Wait for .5 to 4 seconds before playing attacking animation, repeat.
  float randomTime = Random.Range(.5f, 4f);
  yield return new WaitForSeconds(randomTime);
}
```

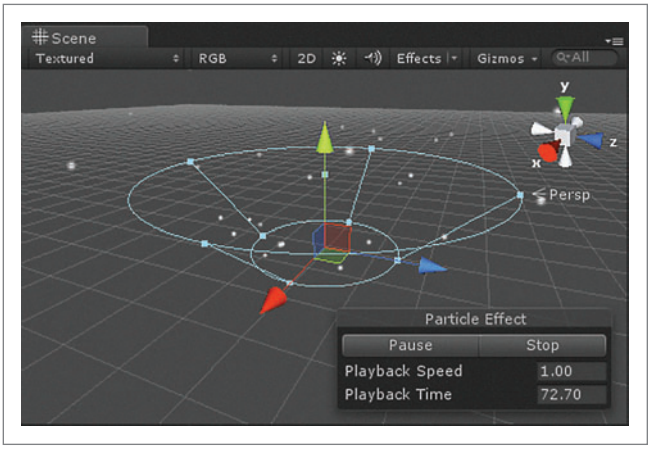


Figure 8 **A Particle Effect**

Physics and Collision Detection

Physics and collision detection features in 3D are nearly the same as in 2D, except the colliders are shaped differently and the rigidbody component has a few different properties, such as being able to accomplish free rotations or movement in the X, Y and Z axes. In 3D there's now a mesh collider that wraps the entire shape of a model as a collision-detection zone. This might sound great, and for collisions it's pretty good, but it's not good for performance. Ideally, you want to simplify collider shapes and limit the processing power it takes to use them. Have a zombie? No problem, use a capsule collider. A complex object? Use multiple colliders. Avoid the mesh collider if possible.

Unity provides a number of methods to know when a collision happens or a trigger is triggered. The following shows just a basic example:

```
void OnCollisionEnter(Collision collision)
{
  // Called when you have a physical collision.
  Debug.Log("Collided with " + collision.gameObject.name);
}

void OnTriggerEnter(Collider collider)
{
  // Called when another object comes within the trigger zone.
  Debug.Log("Triggered by " + collider.gameObject.name);
}
```

There are many more methods than listed here, such as OnTriggerExit and OnCollisionExit and they're almost identical to their 2D counterparts.

Object Creation

When you want to create new GameObject-based items at run time, you don't use constructors. Instead, you use Instantiate. You can certainly have classes with constructors, just not directly in scripts inheriting from MonoBehavior, which happens to be all the top-level scripts assigned to any GameObject. Those scripts can, however, call constructors for other objects all they want:

```
// Assume this reference has been assigned in the editor.
[SerializeField]
private GameObject _zombie;

void Start()
{
  // Create a new instance of that game object. This can be
  // a prefab from your project or object already in scene.
  Instantiate(zombie, transform.position, Quaternion.identity);
}
```

Particle Effects

If you want flashing stars, dust, snow, explosions, fire, mist from a waterfall, blood effects or a number of other effects, you use a particle effect. There's an old particle system in Unity and a newer, more optimized one called Shuriken. You can do so many amazing things with Shuriken in Unity, including having your falling particles support collisions. Because there are many tutorials out there, such

Figure 9 **The Exposed SmokeEffect Property**

```
[SerializeField]
private ParticleSystem _smokeEffect;

void OnTriggerEnter(Collider collider)
{
  // Ensure you only show particles if the player comes within your zone.
  if (collider.gameObject.tag == "Player")
  {
    GameController.Score++;
    // Create particle system at the game objects position
    // with no rotation.
    Instantiate(_smokeEffect, transform.position, Quaternion.identity);
    // Don't do: Destroy(this) because "this"
    // is a script component on a game object, so use
    // this.gameObject, that is, just gameObject.
    Destroy(gameObject);
  }
}
```

as the one at bit.ly/1pZ71it, and they're typically created in the editor with the designer, here I'll just show how they can be instantiated when, say, a character enters the trigger region of a coin to collect.

To get started with particles, simply go to the Game Object | Particle System menu and you'll immediately see one added to your scene, as in **Figure 8**.

I like to create prefabs (which I covered in the second article) from my particle systems so I can easily reuse them, and I can then easily instantiate them via code by first assigning the script to a game object (assuming it's in a class that derives from MonoBehavior, as all game object script components are) and then, in the editor, dragging a particle effect from my scene or a prefab in my project onto, for example, the exposed SmokeEffect property in **Figure 9**.

Creating a UI

Unity 4.6 added a brand new UI system for creating heads-up displays in game elements using text, panels, widgets and more. Adding text to your game's display is simply a matter of clicking on GameObject | UI | Text and setting the font and the text. If you want to control that later via code to perhaps update a score, you simply use:

```
// Gets the UnityEngine.UI.Text component.
  var score = GetComponent<Text>();
  score.text = "Score:0";
```

If I want an image in my UI, I simply click on GameObject | UI | Image and assign a 2D sprite image to this new component. I can set these values just as with any other game object. I hope you see a pattern by now. To create a simple GUI, create the UI objects via the GameObject | UI menu, set the initial values in the Editor and control them later by getting references to those UI components and setting the values, or even animating the values. I built a basic GUI, shown in **Figure 10**, by creating elements underneath a new Canvas component. The new Unity 4.6 UI system contains a number of basic object types, such as Panel, Button, Text, Image, Slider, Scrollbar, and Toggle, and it's incredibly easy to anchor them, scale them, and drag and drop them to create a UI.

Figure 10 **A UI with an Image and Heads-up Text**

AI in Your Game

It wouldn't be fair not to mention AI, though I won't get into creating AI here (even though the building blocks for it are in the earlier code samples for find/move/rotate). But I'll mention a few options that are available to you. I hesitate to call AI in a game AI, because it's not so much intelligence as just a very basic action. I showed you how to have a transform rotate toward another object and move that object. That's the basic AI in many games. Unity has some built-in path-finding capabilities with its NavMesh support, which calculates ahead of time all the paths around objects. NavMesh works pretty well and is now included in the free edition of Unity, although many choose instead to use the A* Pathfinding Project (arongranberg.com/astar), which is an algorithm you can either implement yourself, or save yourself the time by purchasing an asset package for it. As of this writing, 2D path-finding support isn't built into Unity, only 3D, although A* does have that capability. Behave 2.0 from AngryAnt is a popular AI plug-in for Unity with some really strong features, and there's also RAIN, a free AI toolkit from rivaltheory.com, which is also pretty decent and has built-in behaviors for follow, find, Mecanim integration and more.

Wrapping Up

The 3D world adds an extra layer of complexity over 2D as it deals with full meshes and one more dimension. The Asset Store is absolute key for beginners and advanced alike, and you can really get off to a quick start by using pre-created assets.

When I started developing games, I went crazy finding so many models and textures on the Internet. There are some great asset marketplaces out there, but you'll quickly find they aren't all good for games. I once downloaded a small boulder that had near 100,000 vertices in its model! Look for assets that are mobile-optimized, or check out the vertex/polygon count to ensure you find ones that can work for your games. Otherwise, they can slow your performance down considerably. There are optimization tools you can use on models, including one for Unity called Cruncher. In the next article, I'll discuss how to take a game or app from Unity over to the Windows platform. Check out my Channel 9 blog (aka.ms/AdamChannel9) for some videos and links to content to download. ■

Adam Tuliper *is a senior technical evangelist with Microsoft living in sunny Southern California. He's an indie game dev, co-admin of the Orange County Unity Meetup and a Pluralsight.com author. He and his wife are about to have their third child, so reach out to him while he still has a spare moment at adamt@microsoft.com or on Twitter at twitter.com/AdamTuliper.*

Thanks to the following technical experts for reviewing this article: *Matt Newman (Subscience Studios) and Tautvydas Žilys (Unity)*

Use Updatable Tables for Responsive Real-Time Reporting

Don Mackenzie

Business systems often include challenging reporting requirements. A business user needs access to historic transactions and current activity. He also needs to view data in many ways. A user will make specific requests such as:

- Dollars by month by customer for the last year
- Units by week by product for the last six months
- Units and dollars by order and product for the last 10 days (including today and right now)

Satisfying these varied requests can be a challenge for system designers, especially for businesses with a high transaction volume. Cox Digital Solutions (CDS) is a good example. The company processes about 20,000 transactions per second.

My system design for CDS supports up-to-the-hour reporting with a year's history in one SQL Server database table with updates every 10 minutes. The solution blends two SQL Server features—partitioning and columnstore indexing—to achieve this response time when querying such a massive amount of data.

This article discusses:

- Managing massive SQL Server data tables
- Using columnstore indexes and partitioning together
- Switching partitions to facilitate live updates

Technologies discussed:

SQL Server 2012, SQL Server 2014, Hadoop

First Impressions

CDS provides Internet advertising services. "Free" television shows are supported by advertising and "free" Web sites are supported by the ads displayed on those pages. CDS helps Web site publishers display ads properly. The company records information about every display of every ad (called an impression). It also records other information such as clicks on ads. This generates nearly 2 billion records (1.5TB compressed) per day.

Each impression, or transaction, involves numerous components. There are two customers—the advertiser and the Web site publisher. It also involves the product, which is the ad. Advertisers place orders, so there's also an order id as part of a record. There's a price collected from the advertiser and a price paid to the publisher. The transaction record contains many fields that reflect numerous small details about the transaction.

More than a year's transactions are archived so a user can compare the present month to a year ago, or Thanksgiving (a big advertising day) this year to last year. CDS generates reports to bill advertisers and pay publishers. These reports also help the staff monitor advertising-order delivery, they help advertisers monitor the effectiveness of their ads, and the publishers can track activity and revenue.

The business users often can't wait for each new day's data. They'll make adjustments and updates as ads are delivered throughout the day. This led me to the design I'll describe here, which you can adapt for many applications where a continuous stream of new data is added to an already large dataset.

Data Overload

You wouldn't want to put 500 billion rows (and 500TB decompressed) in a SQL Server table. The raw transactions are kept in a Hadoop Distributed File System (HDFS). Hadoop is a good tool for storing and analyzing massive amounts of data, but has terrible response time for queries (learn more at hadoop.apache.org).

> You wouldn't want to put 500 billion rows in a SQL Server table.

The queries against this data take from five minutes to 20 hours, depending, of course, on the amount of data, the time window and the complexity of the query. Making queries directly against HDFS is too slow for the Web application users.

Most users' needs can be met with aggregated data. Transactions are summarized by hour (and by advertiser, Web site, ad and other key attributes) and placed in a SQL Server database table. The hours are important because advertising activity is quite different at noon than at 2 a.m. This process summarizes 70 million transactions to 50,000 rows per hour.

These rows are placed in a SQL Server table called Revenue. An interactive Web application lets the user query this table to check on the performance of an order, see how an ad performs on different sites, review budgets and so on. This table is updated and an hour's data (50,000 rows) is replaced every 10 minutes.

Organizing a Large Database Table

That's still a massive table, having 500 million rows and consuming 60GB of storage. An ordinary table that large would be slow to update and slow to query. Adding indexes can help, but they further slow updates. As tables become larger, it creates multiples layers of indexes (index depth), which reduces their helpfulness.

Multidimensional OLAP Cubes are one approach to querying large amounts of data, but they require lots of design and planning before deployment. And once deployed, you can't update them. OLAP Cubes are typically rebuilt daily, or even less frequently.

My approach to managing this massive table combines two techniques: partitioning and columnstore indexing. A significant limitation of columnstore indexed tables in SQL Server 2012 is that you can't update them while the index is active. You must disable the index during update and completely rebuild it after the update. That's a cumbersome process when dealing with a large table. I'll describe a technique for overcoming this limitation. These restrictions have been relaxed in SQL Server 2014 (see "Partitioning a SQL Server Table").

> My approach to managing this massive table combines two techniques: partitioning and columnstore indexing.

The various partition sizes—month and day and hour—match the common query patterns of the business users. They also match the common update and maintenance patterns. Your application may need different partition sizes and values other than "date" to be most useful for partitioning. In other applications, it might be better to keep just hourly or daily partitions. SQL Server 2012 supports up to 15,000 partitions per table, allowing hourly partitions for 20 months. Refer to the SQL Server documentation at bit.ly/1mtZkfI for more authoritative and in-depth explanation of partitioning.

Columnstore Indexes

The most exciting new feature SQL Server 2012 introduced is columnstore indexes. They're a powerful tool for improving query performance. For massive tables like mine, I've seen a 100x performance increase.

The serious issue in SQL Server 2012 is that a columnstore indexed table, like an OLAP Cube, is read-only. You can't update it

Partitioning a SQL Server Table

SQL Server 2005 introduced a good level support for partitioned tables, which has continued with subsequent versions. Partitioning breaks a large table into several (or many) smaller internal tables. SQL Server stores and indexes each small table, or partition, separately (see **Figure A**).

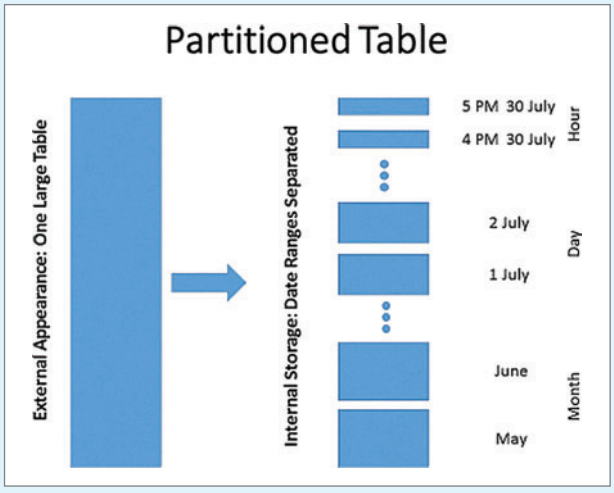


Figure A **Partitioned Table in SQL Server**

A Partition Function (essentially a list of boundary values that separate partitions) tells SQL Server how to separate the table into partitions. There's often a datetime column (as in my Revenue table) that's useful for separating partitions. The table has monthly partitions for data more than a month old, daily partitions for newer data more than a week old, and hourly partitions for very recent data. All these small partitions have a single table name, a single schema and appear to be one large table to the application.

When SQL Server processes a query with a WHERE clause including the date column, it determines which partitions are needed and ignores the rest. All of the queries include the date in the WHERE clause. This improves query performance by restricting the query to one or a few smaller data stores.

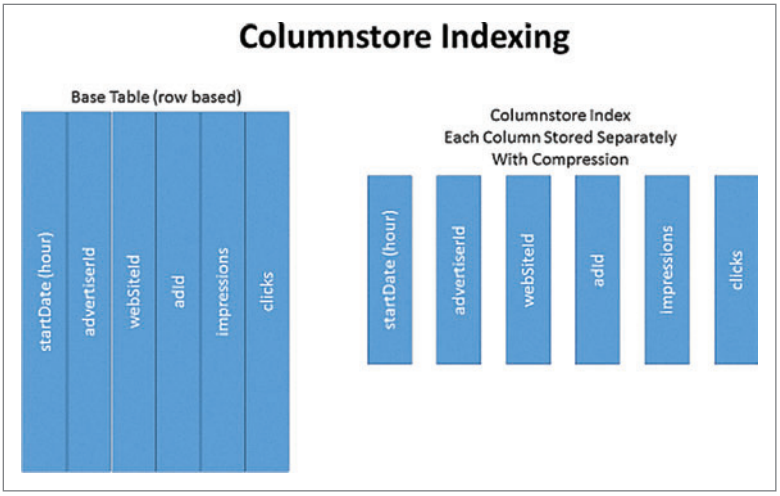


Figure 1 **Use Columnstore Indexes to Manage Large Data Stores**

without dropping (or at least disabling) the index and rebuilding it after the update. After verifying the performance improvements, I looked for a way to overcome the read-only limitation and found partitioning was the answer.

> I think of columnstore indexes as slicing a table into vertical slices by column.

While I think of partitioning as slicing a table into horizontal slices (based on time in my case), I think of columnstore indexes as slicing a table into vertical slices by column. Like all Nonclustered indexes, the index is stored separately from the main table data. Each columnstore indexed column is stored individually in its own internal blob container. This is quite different from a composite index with multiple indexed or included columns together.

Because the base table is read-only, the data in the columnstore blobs don't need to handle updates and they can be compressed using one of several algorithms chosen by SQL Server to match the data. So the index is often substantially smaller than the original data (see **Figure 1**).

My columnstore index includes every column in the table. Each column is stored separately. A query that references only a few columns will only read those columns and ignores the rest. Queries don't need to read the rows in the base table. All the data values are in the index and the data for each column in the index is compressed to minimize disk reads.

Remember my table is also partitioned, so each partition and each column in the partition has its own storage (see **Figure 2**). SQL Server is smart enough to only read the data it needs (partitions and columns) for a specific query with impressive query-response time improvement compared to scanning a "normal" table row by row, or even searching through normal indexes for needed data.

An important positive side effect of all this is there's no performance penalty for denormalizing the table. The former implementation suffered because queries against the precursor to the Revenue table often included JOINs to other tables to get secondary attributes. At that time, I wanted to keep the table rows skinny and avoid reading unnecessary data from disk every time the large table was queried.

Sometimes the JOINs caused SQL Server to do a full table scan for a query, especially when the secondary attribute was in the WHERE clause. Now, many of those secondary (and some tertiary) attributes are stored in the much wider Revenue table knowing that with columnstore indexing, they won't be accessed unless they're needed by that specific query. The Revenue table I've designed has 25 columns. Refer to the SQL Server documentation at bit.ly/1zbsju1 for more authoritative and in-depth understanding of columnstore indexes.

Updating

The combination of partitioning and columnstore indexing has broken the massive table into manageable modest-sized segments.

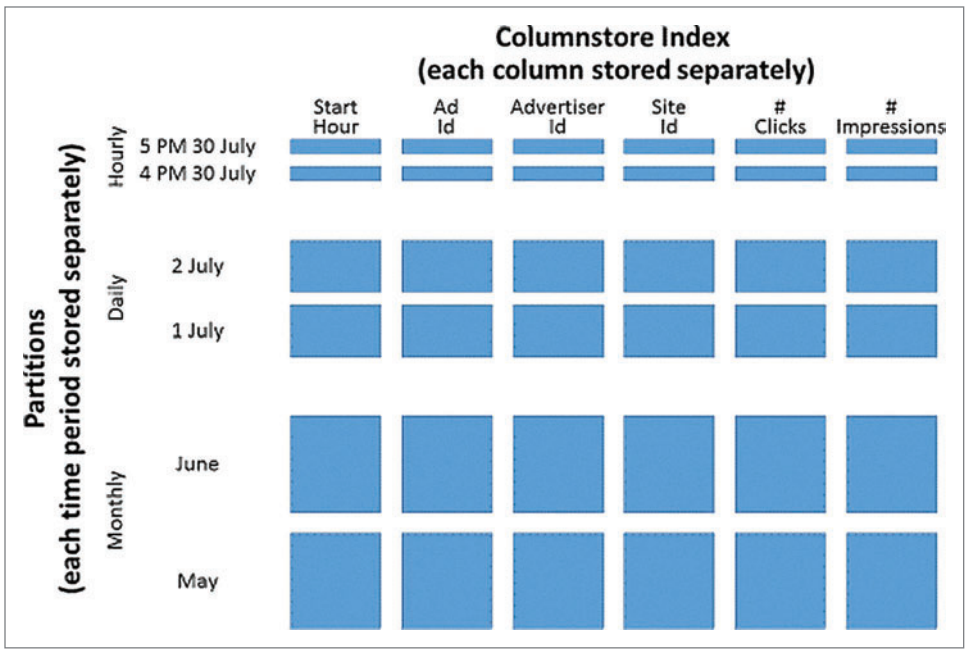
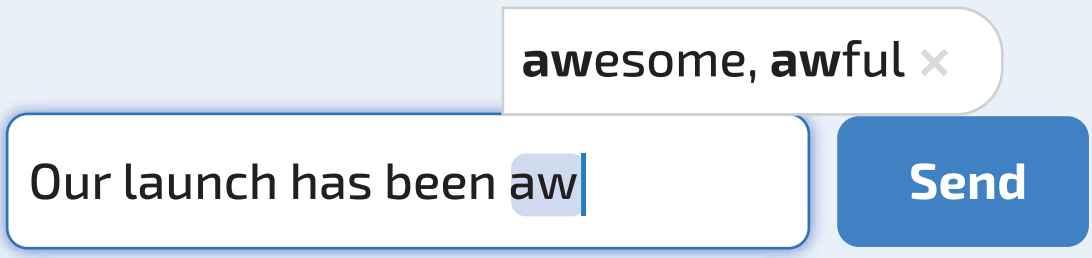


Figure 2 **Using Partitions and Columnstore Indexes to Parse Data**



toptal

Exclusive access to top developers.

www.toptal.com

However, because of the columnstore index, updates are forbidden in SQL Server 2012. Even though each partition is stored separately, they're all treated as one table, so disabling, dropping or rebuilding the index involves the whole table. Rebuilding the columnstore index can take a half hour and the table isn't available for queries during that time. The table needs to be updated with new data every 10 minutes and be available all the time.

The key to solving this problem is a partition-related statement: ALTER TABLE ... SWITCH PARTITION ... This DDL statement moves a partition of data from one table to another. It doesn't copy the data, but just rearranges internal schema information so the partition storage that used to belong to one table now belongs to a different table. There are some rules governing this process, but they're manageable. Read the TechNet Library article, "Transferring Data Efficiently by Using Partition Switching" (bit.ly/1ts04Xv), for a thorough description of partition switching.

> A user has the luxury of fresh, current data and more than a year's history at his fingertips in one table that's easy and fast to query.

Let's revisit the original problem. There's a torrent of data (20,000 transactions/second) from the ad serving machines going to Hadoop 24x7. I can use Hadoop to summarize the current hour's data to about 50,000 SQL Server rows for that one hour. That Hadoop query takes about 5 minutes. I can't update my main Revenue table, but I can put those rows into a new table (partitioned and columnstore indexed) I call RevenueIn. It has the same schema as the Revenue table, but it's empty and its columnstore index is disabled, so I can insert rows.

After inserting the hour's rows, I can rebuild the columnstore index on the RevenueIn table. The hour's data fits precisely into one partition. I'm only indexing 50,000 rows, so that takes less than a minute. Now I can use SWITCH PARTITION to move that one partition, already indexed, into the main Revenue table and it's instantly available for report queries. Note that the SWITCH PARTITION statement only works if the destination partition is empty. To solve that problem, I use a third table called RevenueOut, which is empty.

Existing data from the Revenue partition is SWITCHed to RevenueOut (leaving the Revenue partition empty) and then the RevenueIn partition (with the new data and its index) is SWITCHed into Revenue. The two SWITCH statements take less than 5 ms in my environment. The old data in RevenueOut is then truncated (see **Figure 3**). This is a variation on the sliding window partition problem described at bit.ly/1wgPVkR.

Details, Details

There are always details. The application repeats this process six times or more per hour (as fast as the Hadoop query can run). Early in the hour, there are only a few minutes of data and each update during the hour replaces that with fresh data. Shortly after the top of the hour, there's a full hour of data in the huge Revenue table and it can begin on the next hour. The user has the luxury of fresh, current data and more than a year's history at his fingertips in one table that's easy and fast to query.

Developers also win with a single database table. The previous solution used one table for "current" data and another for historic data. C# code had to decide which tables to use and combine the results of database queries if a query used both tables. The single table and minimal use of JOINs (because of the denormalized columns enabled without penalty by columnstore indexing) simplifies the database queries for reporting.

Another detail is merging. I keep older data in monthly partitions. This simplifies maintenance as I can drop a month that's beyond the requirements. Fewer partitions also simplify the SQL Server internal logic to select partitions for queries, as most queries for old data include whole months. The columnstore index also gains some efficiency with larger partitions due to its compression algorithms. I use a technique similar to inserting data for rolling up small partitions into large partitions without rebuilding the data.

There is some SQL Server plumbing involved. An accompanying online article at msdn.microsoft.com/magazine/dn800596 provides step-by-step instructions and sample T-SQL for all the necessary SQL Server database objects. Check out that article and copy the code from there. ■

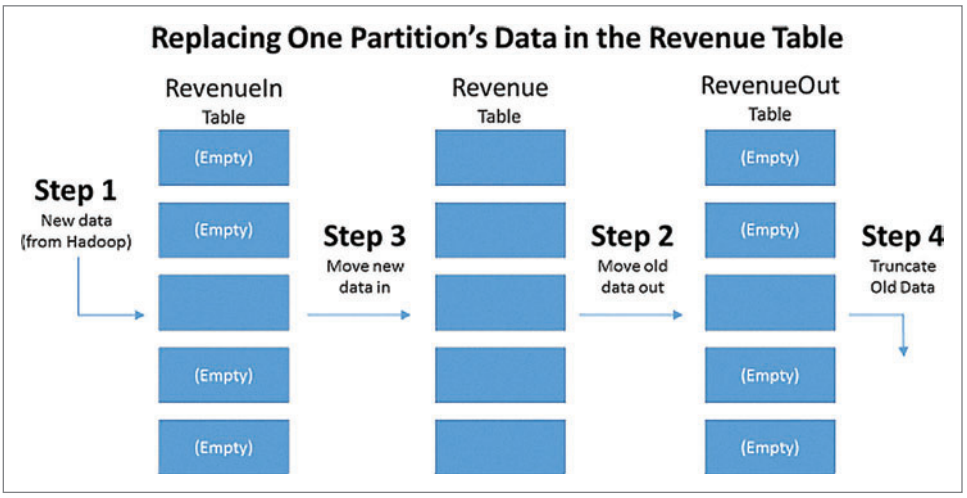


Figure 3 **Refreshing a Partition's Data**

Don Mackenzie *is director of software architecture at Cox Digital Solutions, the Internet advertising arm of Cox Media Group and Cox Enterprises. He enjoys applying new technologies to business applications. Reach him at don@coxds.com.*

Thanks to the following Microsoft technical expert for reviewing this article: *Remus Rusanu*


Deploy deep
into space with
InstallAware 18
› New InstallAware Direct Deploy Technology. Pushes any EXE without ANY client or server software.
› Complete Unicode support, retaining full backwards compatibility with previous InstallAware projects.
› Bullet-proof, depenency free; run the same setup on Windows XP all the way to Windows Server 2012 R2 x64!
› Supports all Active Directory Domains.
› Competitive upgrade discounts from InstallShield.
www.installaware.com
InstallAware

Reuse Code Assets with Universal Windows Apps

Joel Reyes

A lot has changed since I wrote my July 2013 article, "Building Apps for Windows 8 and Windows Phone 8" (msdn.microsoft.com/magazine/dn296517). Today, developers with experience building apps for Windows 8.1 can reuse code and share assets to build new experiences for Windows Phone 8.1. Windows Phone 8 developers can also maximize the same code base when targeting Windows Phone 8.1. Visual Studio 2013, together with a unified Windows Runtime, makes it possible to develop apps that target Windows Phone 8.1 and other Windows devices from one Visual Studio solution. This new approach allows developers to significantly broaden the user base of their apps while maximizing code reuse.

In this article, I'll share information that will help you get up to speed on the new universal Windows app capabilities found in Visual Studio 2013, including templates, languages, APIs, compilers, different approaches for code sharing, and considerations when updating platform-specific apps to universal Windows apps.

Figure 1 shows the relationship between the universal Windows apps template, the app solution and the compilation output package.

This article discusses:

- Universal app templates in Visual Studio 2013
- Project, language, API, compiler, code and XAML editor support
- Code-sharing strategies
- Application Update Paths

Technologies discussed:

Visual Studio 2013, Windows 8.1, Windows Phone 8.1, XAML

Using the templates included with Visual Studio, such as the Hub app project template, you can create solutions structured to allow separate projects for each platform, as well as a special project for sharing. This shared space is where you can define the code to be reused for the applications targeting PCs, tablets and Windows Phone. This structure ensures all the elements unique to each app are within the appropriate project. The build process creates each specific application package, integrating all dependencies from the shared project. The Hub app template is a great starting point for building shopping, news, sports and media apps.

Figure 2 shows the Universal App New Project Hub App template for C#.

Starting with Update 2, Visual Studio included all the necessary support to make it easy to build Universal Windows apps that maximize code reuse.

App Project Support

In a typical scenario where developers target both Windows 8.1 and Windows Phone 8.1, Solution Explorer displays three distinct projects, as shown in **Figure 3**:

1. A Windows project (Windows 8.1): for XAML pages and specific code for Windows.
2. A Windows Phone project (Windows Phone 8.1): for XAML pages and specific code for Windows Phone.
3. A Shared project: for any assets sharable between the target apps, such as .cs, .xml .png, .resw and other items. The Share project imports automatically into each platform project during the build process.

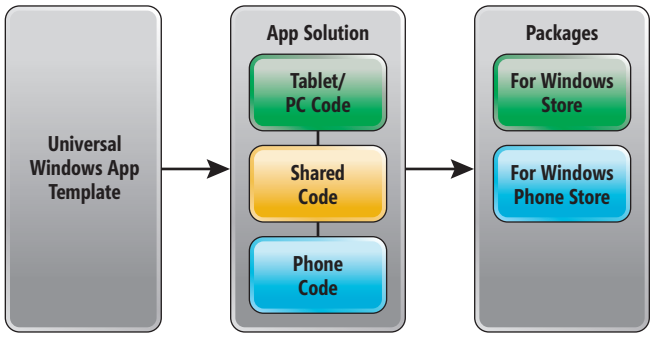


Figure 1 **The Universal Windows App Template, Solution and Output Packages**

For existing solutions you can use Add Windows Phone 8.1 or Add Windows 8.1 from the project context menu.

566 classes, 119 struts and 57 interfaces. There are, however, some APIs that are available only in Windows Phone.

Compiler Support

Although the build process guarantees importing the associated XAML, code and assets for each project, sometimes you might need to include platform-specific code in the Shared Project. You can leverage the conditional compilation directives #if and #endif to indicate the code intended for each platform. For C#, the preprocessor compiler symbols are WINDOWS_APP and WINDOWS_PHONE_APP; for C++, they are WINAPI_FAMILY_PC_APP and WINAPI_FAMILY_PHONE_APP.

The following C# example illustrates conditional compilation for a situation where the developer may want to move the App.xaml file to the Shared Project (this example assumes any styles defined in it use resources available to other types of apps):

```
#if WINDOWS_APP
  if (!rootFrame.Navigate(typeof(HubPage)))
#endif
#if WINDOWS_PHONE_APP
  if (!rootFrame.Navigate(typeof(WindowsPhoneStartPage)))
#endif
  // ...throw exception
```

Here's the equivalent C++ conditional compilation:

```
#if WINAPI_FAMILY==WINAPI_FAMILY_PHONE_APP
  if (!rootFrame->Navigate(WindowsPhoneStartPage::typeid, e->Arguments))
#else
  if (!rootFrame->Navigate(HubPage::typeid, e->Arguments))
#endif
  // ...throw exception
```

Because universal Windows apps run on the same Windows Runtime, developers have a common way to architect and build apps for the range of Windows devices, from how they handle suspend and resume and do background processing, to the way they manage in-app security.

Language Support

Code sharing can be accomplished using C++, C#, or JavaScript, whether your design involves XAML, HTML, or DirectX. For Visual Basic, recent updates allow calls to Windows Runtime (WinRT) APIs and include support for XAML and the assets folder of the Portable Class Library (PCL). F# developers can create shared code for both legacy and newer Microsoft .NET Framework and Windows Store apps.

API Support

A common Windows Runtime means massive unification of the supporting APIs for both platforms, Windows 8.1 (for PCs and tablets) and Windows Phone 8.1. This unification is the key to easier and more capable cross-platform development. Because universal Windows apps run on the same Windows Runtime, developers have a common way to architect and build apps for the range of Windows devices, from how they handle suspend and resume and do background processing, to the way they manage in-app security. In fact, in terms of key language constructs, the Windows 8.1 SDK and Windows Phone 8.1 SDK share

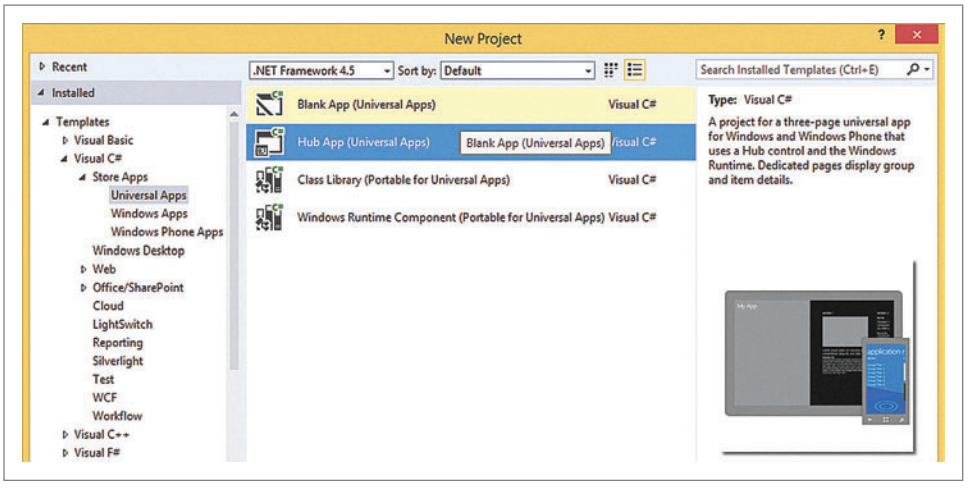


Figure 2 **Universal Windows App New Project Hub App Template**

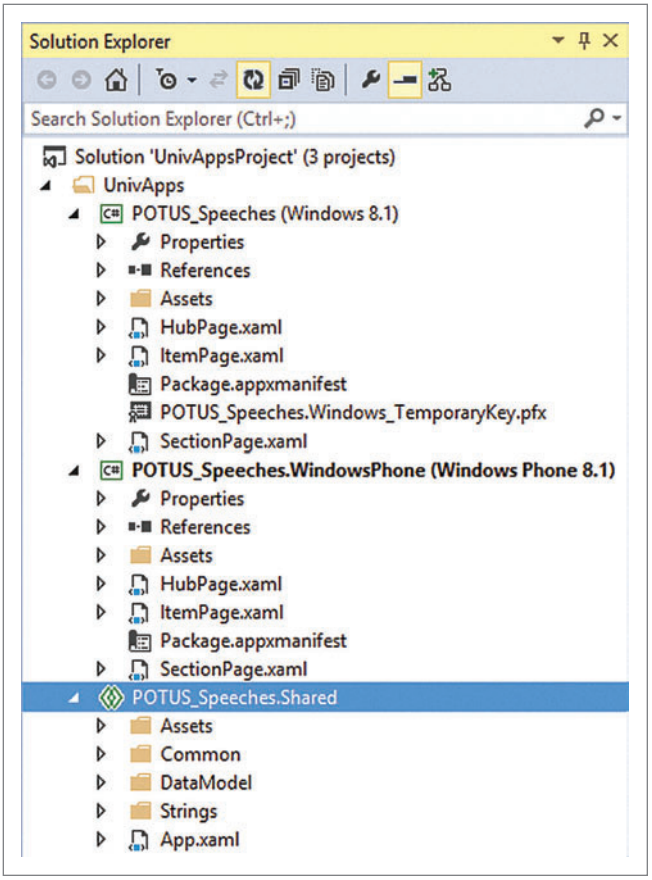


Figure 3 **Universal Windows App Projects in Solution Explorer**

Code and XAML Editor Support

Visual Studio provides a lot of help to ease the development of universal apps. IntelliSense monitors your code to prevent the use of unsupported APIs in your Universal App project. Corresponding error messages will identify an inappropriate API when you build the project, as shown in **Figure 4**.

The navigation bar allows you to switch project context. This is useful when you want to concentrate on only one project in Solution Explorer. The project context (Windows Phone or Windows 8) drives the IntelliSense experience in the editor and the designer.

During debugging, you may want to specify the default startup project for F5. Visual Studio provides an easy way to do this by using the debug target dropdown and, under Startup project, selecting the desired project. This feature is supported in both stores and is turned on when there's more than one application project in the solution.

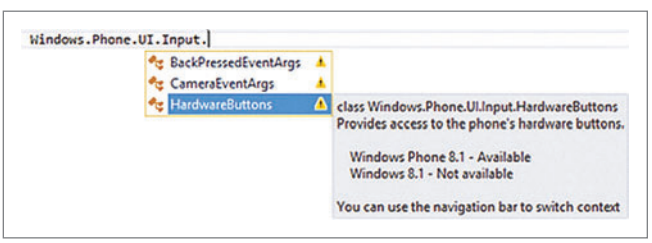


Figure 4 **IntelliSense Showing an API Usage Error**

The Device Panel, shown in **Figure 5**, helps you make sure your application behaves as expected on all target devices. It lets you test your application using different display resolutions, orientations, accents, themes and so forth.

Code Sharing Strategies

When building universal Windows apps, it's best to maximize the application artifacts sharable across apps. Traditionally, developers have relied on linked files to share code across multiple projects. Visual Studio 2013 enhances this capability and provides two related artifacts to enable code sharing: the Shared Project and PCL.

A Shared Project (Shared Folder) offers a simple organizational structure for storing all the shared code and assets, including app.xaml and other XAML files, code files, images, XML/JSON and .resw files, as well as templates such as Common and DataModel. References in the Shared Project need to be resolved to resolve compiler errors.

The PCL is better supported in universal Windows apps. For instance, developers can now call and create WinRT APIs and portable XAML user controls. When you change the project target to include another platform, the compiler produces a single binary guaranteed to work on all the targets. You can also make a PCL target a specific platform by removing the other targets. This in-place conversion is only supported between PCL, Windows Store 8.1 and Windows Phone 8.1.

These PCL binaries can also be distributed to third parties, and this has generated a growing community of cross-platform developers who are porting their libraries to PCL. This ensures that abundant library capabilities will be available to developers, including Visual Studio Express developers, for targeting other platforms. Through MS Open Tech, Microsoft works with various open source communities to contribute code to popular frameworks and optimize them for Windows devices. For instance, JQuery fully supports the Windows Runtime so developers can reuse their existing code and skills to build Windows 8.1 apps.

Deciding Between Shared Projects and PCL

There's a simple determinant for choosing between Shared Projects and PCLs: platform specificity.

- If your project isn't targeting third-party distribution, Shared Projects will give you the best experience.
- If your project involves third-party distribution, PCL is the better choice.

Figure 6 indicates some of the strengths and weaknesses of each approach.

Other Code Sharing Considerations

Linking Files: Similar to Shared files, the Visual Studio Add as Link feature gives you explicit file sharing the traditional way, using compiler conditionals to share across additional platforms.

Code Sharing in the Front End: Both Windows Store 8.1 and Windows Phone 8.1 share the same WinRT XAML stack. This is great news as it allows developers to also target the UI for sharing. You can share the UI via a PCL, including using the XAML designer.

Code Sharing in the Back End: Universal Windows apps fully benefit from code sharing focused on back-end processing, and this

SharePoint LIVE!
TRAINING FOR COLLABORATION

ORLANDO NOV 17-21
ROYAL PACIFIC RESORT AT UNIVERSAL ORLANDO

LIVE! 360
TECH EVENTS WITH PERSPECTIVE

5 GREAT CONFERENCES. 1 GREAT PRICE.

- SharePoint LIVE! — TRAINING FOR COLLABORATION
- Visual Studio LIVE! — EXPERT SOLUTIONS FOR .NET DEVELOPERS
- SQL Server LIVE! — TRAINING FOR DBAs AND IT PROS
- Modern Apps LIVE! — MOBILE, CROSS-DEVICE & CLOUD DEVELOPMENT
- TECHMENTOR — IN-DEPTH TRAINING FOR IT PROS

live360events.com

USS COLLABORATE: LET'S WORK TOGETHER

Climb aboard the USS Collaborate, where SharePoint Live! will provide leading-edge knowledge and training for SharePoint administrators, developers, and planners who must customize, deploy and maintain SharePoint Server and SharePoint Foundation to maximize the business value.

splive360.com

REGISTER BY OCTOBER 15 AND SAVE $300

Use promo code SPOCT1

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

EVENT SPONSORS: Magenic, Microsoft
PLATINUM SPONSOR: esri
GOLD SPONSORS: New Relic, sitecore Own the experience
PRODUCED BY: 1105 MEDIA

is perfect for implementing shared data models with cloud dependencies. Microsoft Azure Mobile Services, in particular, is a good example of powerful back-end components that universal Windows apps can leverage across applications and devices to access services for notification and other functions. It serves both Windows Store and Windows Phone applications equally well, with the same level of data security, application availability and scalability.

Application Update Paths

Developers with apps in either store can now easily update the app to support the other store. All update paths require you to have Windows 8.1 and Visual Studio 2013 installed.

Updating a Windows Phone 8 app to a universal Windows app: Updating your Windows Phone 8 app to the Windows Runtime as a universal Windows app means you can use Windows XAML to create the new UX, and offer features like notification options that capitalize on the new Windows Phone 8.1 Action Center, or even offer entirely new features to users. Once you've updated to a universal app, you can easily share code with a parallel app for PCs and tablets.

If you're updating your app from Windows Phone 8 to Windows Phone 8.1, check out the documentation for differences in API support, as there are several API behavior and feature changes (aka.ms/waphone81) related to background processing, files and storage, controls, and other device capabilities.

Updating a Windows Store app to a universal Windows app: If you've already built a Windows 8.1 app and want to publish to the Windows Phone 8.1 store, the universal Windows apps pattern also applies. As in the case of Windows Phone 8.1, it's important to understand the API behavior changes (aka.ms/wawindows81) that may affect the migration process. For example, the AppBar, Flyout, Hub, and other controls are not supported on Windows Phone.

Updating a Windows Store app and adding Windows Phone 8.1 app: Providing a companion Windows Phone 8.1 app to your Windows Store app is simplified by the universal Windows app template, which allows you to simply add a new Windows Phone 8.1 project to your Windows Store solution. The code editor and compiler recognize the addition and make the appropriate changes to behaviors to guarantee successful development.

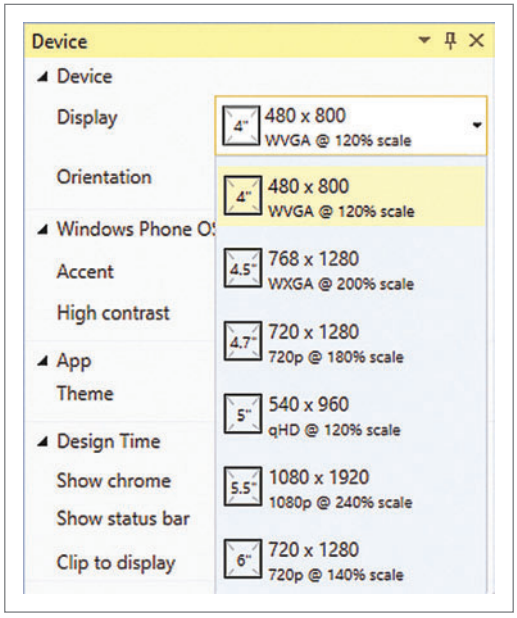


Figure 5 **Device Panel**

Figure 6 **Comparison of Shared Project and Portable Class Library**

	Strengths	Weaknesses
Shared Project	• Allows the use of platform-specific APIs by using #if. • Can handle cases where the source is compatible, but the binary wouldn't be, as when APIs are in different namespaces.	• Requires sharing either the source or multiple binaries, which forwards part of the complexity to the consumer. • Requires convention and discipline to centralize handling of divergent APIs so that the code doesn't become unmaintainable.
Portable Class Library	• Extremely simple deployment and sharing model. • Scales well to large systems that require multiple solutions, involving third parties or multiple programming languages.	• Relies heavily on the fact that the targets have converged APIs. • Handling divergent APIs requires using higher-level abstracts, such as dependency injection or Inversion of Control containers.

Store Certification

The Windows App Certification Kit (WACK) is an instrumental tool for developers preparing their application for submission to the store. You can also use the tool to test your Windows Phone 8.1 apps, but not Windows Phone 8 apps. WACK 3.3 is included in the Windows SDK for Windows 8.1 and you can find more information about the new requirements for WACK at bit.ly/1qXPKmf.

Wrapping Up

Platform convergence has been a long time coming and although we haven't reached nirvana, the progress has given developers substantial improvements in platform capabilities, tools enhancement, UX features, and breadth of application scenarios support. Today, it isn't enough to be able to execute code in different devices. The UX must maximize the specific characteristic of the device on which the application runs. It's equally important to support feature parity across those devices. It's simply what users expect.

This article has explored the key capabilities that allow you to simultaneously develop apps for both the PC/tablet and Windows Phone platforms, whether building solutions from scratch or upgrading from existing ones. Universal Windows apps offer an excellent way for developers to increase the value of their code investment, while broadening users' reach.

You can find a comprehensive assortment of universal Windows app samples that you can execute on both Windows 8.1 and Windows Phone 8.1 at aka.ms/wasamples, and a large collection of feature samples that use the new universal Windows app template at aka.ms/uap. ■

JOEL REYES *is senior technology evangelist in the Microsoft Developer Experience & Evangelism Group focused on Startup Development Strategy. You can reach him at joel.reyes@microsoft.com.*

THANKS to the following Microsoft technical expert for reviewing this article: *Lora Heiny*

Modern Apps LIVE!

MOBILE, CROSS-DEVICE & CLOUD DEVELOPMENT

Presented in partnership with Magenic

ORLANDO NOV 17-21

ROYAL PACIFIC RESORT AT UNIVERSAL ORLANDO

LIVE! 360

TECH EVENTS WITH PERSPECTIVE

5 GREAT CONFERENCES. 1 GREAT PRICE.

- Modern Apps LIVE! — MOBILE, CROSS-DEVICE & CLOUD DEVELOPMENT
- Visual Studio LIVE! — EXPERT SOLUTIONS FOR .NET DEVELOPERS
- SharePoint LIVE! — TRAINING FOR COLLABORATION
- SQL Server LIVE! — TRAINING FOR DBAs AND IT PROS
- TECHMENTOR — IN-DEPTH TRAINING FOR IT PROS

live360events.com

MASTER THE MODERN APPS LANDSCAPE

Presented in partnership with Magenic, Modern Apps Live! brings Development Managers, Software Architects and Development Leads together to break down the complex landscape of mobile, cross-platform, and cloud development and learn how to architect, design and build a complete Modern Application from start to finish.

modernappslive.com

REGISTER BY OCTOBER 15 AND SAVE $300

Use promo code MALOCT1

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

EVENT PARTNERS: Magenic, Microsoft

PLATINUM SPONSORS: esri

GOLD SPONSORS: New Relic, sitecore Own the experience

PRODUCED BY: 1105 MEDIA

JAMES MCCAFFREY

Probit Classification Using C#

Probit ("probability unit") classification is a machine learning (ML) technique that can be used to make predictions in situations where the dependent variable to predict is binary—that is, it can take one of just two possible values. Probit classification is also called probit regression and probit modeling.

Probit classification is quite similar to logistic regression (LR) classification. The two techniques apply to the same types of problems and tend to give similar results, and the choice of using probit or LR classification usually depends on the discipline in which you're working. Probit is often used in economics and finance, while LR is more common in other fields.

To get an understanding of what probit classification is, take a look at the demo program in **Figure 1**.

> Probit classification is a machine learning technique that can be used to make predictions in situations where the dependent variable to predict is binary.

The demo uses probit classification to create a model that predicts whether a hospital patient will die, based on age, sex and the results of a kidney test. The data is completely artificial. The first raw data item is:

```
48.00   +1.00   4.40   0
```

The raw data consists of 30 items. Sex is encoded as -1 for male and +1 for female. The variable to predict, Died, is in the last column and is encoded as 0 = false (therefore the person survived) and 1 = true. So, the first data item indicates a 48-year-old female with a kidney score of 4.40 who survived. The demo begins by normalizing the age and kidney data so that all values have roughly the same magnitude. The first data item, after normalization, is:

```
-0.74   +1.00   -0.61   0.00
```

Normalized values less than 0.0 (here, both age and kidney score) are below average, and values greater than 0.0 are above average.

Code download available at msdn.microsoft.com/magazine/msdnmag1014.

The source data is then randomly split into a 24-item training set to create the model, and a six-item test set to estimate the accuracy of the model when applied to new data with unknown results.

The demo program then creates a probit model. Behind the scenes, training is performed using a technique called simplex optimization, with the maximum number of iterations set to 100. After training, the weights that define the model are displayed { -4.26, 2.30, -1.29, 3.45 }.

The first weight value, -4.26, is a general constant and doesn't apply to any one specific predictor variable. The second weight,

```
file:///C:/ProbitClassification/bin/Debug/ProbitClassification.EXE

Begin Probit Binary Classification demo
Goal is to predict death (0 = false, 1 = true)

Raw data:

       Age       Sex      Kidney   Died
=======================================
[ 0]   48.00     1.00     4.40     0.00
[ 1]   60.00    -1.00     7.89     1.00
[ 2]   51.00    -1.00     3.48     0.00
[ 3]   66.00    -1.00     8.41     1.00
[ 4]   40.00     1.00     3.05     0.00
. . .
[29]   68.00    -1.00     8.38     1.00

Normalizing age and kidney data
Done

Normalized data:

[ 0]  -0.74     1.00    -0.61     0.00
[ 1]   0.19    -1.00     1.36     1.00
[ 2]  -0.51    -1.00    -1.12     0.00
[ 3]   0.66    -1.00     1.65     1.00
[ 4]  -1.36     1.00    -1.37     0.00
. . .
[29]   0.82    -1.00     1.63     1.00

Creating train (80%) and test (20%) matrices
Done

Normalized training data:

[ 0]  -0.43     1.00    -0.99     0.00
[ 1]  -0.51    -1.00     0.26     0.00
[ 2]   1.44    -1.00     1.31     1.00
. . .
[23]  -0.58    -1.00     0.52     0.00

Creating probit binary classifier
Setting maxEpochs = 100
Starting training
Training complete

Best weights found:
-4.2650 2.3076 -1.2965 3.4582

Prediction accuracy on training data = 1.0000
Prediction accuracy on test data = 0.8333

End probit binary classification demo
```

Figure 1 **Probit Classification in Action**

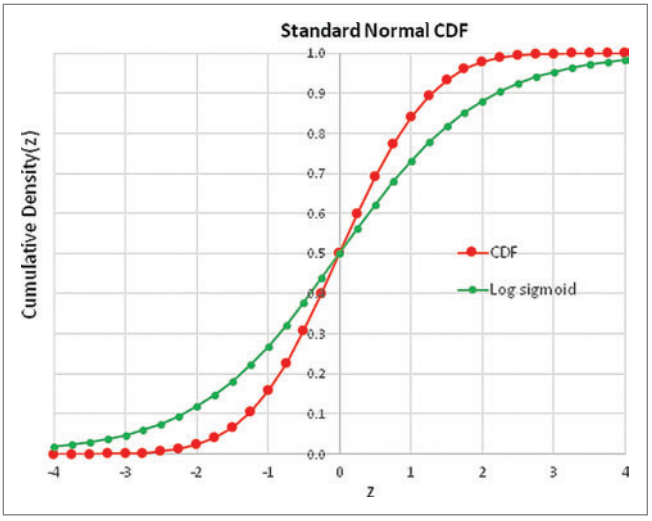


Figure 2 **The Graph of the Cumulative Density Function**

2.30, applies to age; the third weight, -1.29, applies to sex; and the fourth weight, 3.45, applies to kidney score. Positive weights, such as the ones associated with age and kidney score, mean larger values of the predictor indicate the dependent variable, Died, will be larger—that is, closer to true.

The demo computes the accuracy of the model on the 24-item training set (100 percent correct) and on the test set (83.33 percent, or five correct and one wrong). The more significant of these two values is the test accuracy. It's a rough estimate of the overall accuracy of the probit model.

This article assumes you have at least intermediate programming skills and a basic understanding of ML classification, but doesn't assume you know anything about probit classification. The demo program is coded using C#, but you should be able to refactor the demo to other .NET languages without too much trouble. The demo code is too long to present in its entirety, but all the code is available in the download that accompanies this article at msdn.microsoft.com/magazine/msdnmag1014. All normal-error checking has been removed to keep the main ideas clear.

Understanding Probit Classification

A simple way to predict death from age, sex and kidney score would be to form a linear combination along the lines of:

$$died = b_0 + (b_1)(age) + (b_2)(sex) + (b_3)(kidney)$$

where the b_0, b_1, b_2, b_3 are weights that must somehow be determined so the computed output values on the training data closely match the known dependent variable values. Logistic regression extends this idea with a more complicated prediction function:

$$z = b_0 + (b_1)(age) + (b_2)(sex) + (b_3)(kidney)$$
$$died = 1.0 \ / \ (1.0 + e^{-z})$$

The math is very deep, but the prediction function, called the logistic sigmoid function, conveniently always returns a value between 0.0 and 1.0, which can be interpreted as a probability. Probit classification uses a different prediction function:

$$z = b_0 + (b_1)(age) + (b_2)(sex) + (b_3)(kidney)$$
$$died = Phi(z)$$

The Phi(z) function is called the standard normal cumulative density function (which is usually abbreviated CDF) and it always returns a value between 0.0 and 1.0. The CDF is tricky because there's no simple equation for it. The CDF for a value z is the area under the famous bell-shaped curve function (the Gaussian function) from negative infinity to z.

This sounds a lot more complicated than it really is. Take a look at the graph in **Figure 2**. The graph shows the logistic sigmoid function and the CDF function plotted side by side. The important point is that for any value z, even though the underlying functions are very different, both functions return a value between 0.0 and 1.0 that can be interpreted as a probability.

So, from a developer's point of view, the first challenge is to write a function that computes the CDF for a value z. There's no simple equation to compute CDF, but there are dozens of exotic-looking approximations. One of the most common ways to approximate the CDF function is to compute something called the erf function (short for Error Function) using an equation called A&S 7.1.26, and then use erf to determine CDF. Code for the CDF function is presented in **Figure 3**.

To summarize, probit classification uses the CDF function to compute an output value. The CDF function is also called phi. CDF is an area under the bell-shaped curve function and has no simple equation. A common way to approximate CDF is to use formula A&S 7.1.26 to get erf and then use erf to get CDF.

With a CDF function in hand, it's easy to compute the probit output for a set of input values and a set of weight values:

```
public double ComputeOutput(double[] dataItem, double[] weights)
{
  double z = 0.0;
  z += weights[0]; // The b0 constant
  for (int i = 0; i < weights.Length - 1; ++i) // data might include Y
    z += (weights[i + 1] * dataItem[i]); // skip first weight
  return CumDensity(z);
}
```

The second challenge when writing probit classification code is to determine the values for the weights so when presented with training data, the computed output values closely match the known output values. Another way of looking at the problem is that the goal is to minimize the error between computed and known output values. This is called training the model using numerical optimization.

> There's no easy way to train most ML classifiers, including probit classifiers.

There's no easy way to train most ML classifiers, including probit classifiers. There are roughly a dozen major techniques you can use, and each technique has dozens of variations. Common training techniques include simple gradient descent, back-propagation, Newton-Raphson, particle swarm optimization, evolutionary optimization and L-BFGS. The demo program uses one of the oldest and simplest training techniques—simplex optimization.

Understanding Simplex Optimization

Loosely speaking, a simplex is a triangle. The idea behind simplex optimization is to start with three possible solutions (hence,

Figure 3 **The CDF Function in C#**

```
static double CumDensity(double z)
{
  double p = 0.3275911;
  double a1 = 0.254829592;
  double a2 = -0.284496736;
  double a3 = 1.421413741;
  double a4 = -1.453152027;
  double a5 = 1.061405429;

  int sign;
  if (z < 0.0)
    sign = -1;
  else
    sign = 1;

  double x = Math.Abs(z) / Math.Sqrt(2.0);
  double t = 1.0 / (1.0 + p * x);
  double erf = 1.0 - (((((a5 * t + a4) * t) + a3) * t + a2) * t + a1) *
    t * Math.Exp(-x * x);
  return 0.5 * (1.0 + (sign * erf));
}
```

"simplex"). One solution will be "best" (have the smallest error), one will be "worst" (largest error), and the third is called "other." Next, simplex optimization creates three new potential solutions called "expanded," "reflected," and "contracted." Each of these is compared against the current worst solution, and if any of the new candidates is better (smaller error), the worst solution is replaced.

Simplex optimization is illustrated in **Figure 4**. In a simple case where a solution consists of two values, such as (1.23, 4.56), you can think of a solution as a point on the (x, y) plane. The left side of **Figure 4** shows how three new candidate solutions are generated from the current best, worst and "other" solutions.

First, a centroid is computed. The centroid is the average of the best and other solutions. In two dimensions, this is a point halfway between the best and other points. Next, an imaginary line is created, which starts at the worst point and extends through the centroid. The contracted candidate is between the worst and centroid points. The reflected candidate is on the imaginary line, past the centroid. And the expanded candidate is past the reflected point.

In each iteration of simplex optimization, if one of the expanded, reflected or contracted candidates is better than the current worst solution, worst is replaced by that candidate. If none of the three candidates generated are better than the worst solution, the current worst and other solutions are moved toward the best solution to points somewhere between their current position and the best solution, as shown in the right-hand side of **Figure 4**.

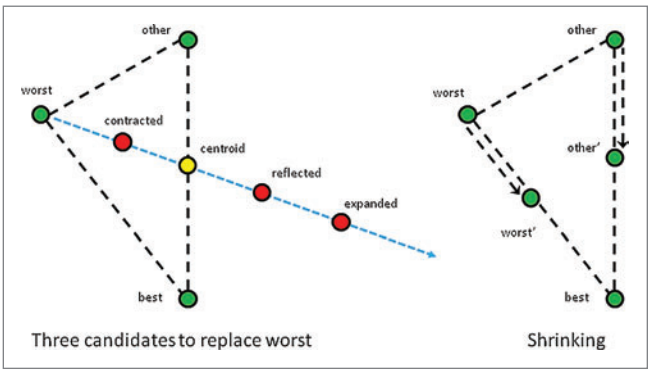


Figure 4 **Simplex Optimization**

After each iteration, a new virtual "best-other-worst" triangle is formed, getting closer and closer to an optimal solution. If a snapshot of each triangle is taken, when looked at sequentially, the shifting triangles resemble a pointy blob moving across the plane in a way that resembles a single-celled amoeba. For this reason, simplex optimization is sometimes called amoeba method optimization.

There are many variations of simplex optimization, which differ in how far the contracted, reflected, and expanded candidate solutions are from the current centroid, and the order in which the candidate solutions are checked to see if they're better than the current worst solution. The most common form of simplex optimization is called the Nelder-Mead algorithm. The demo program uses a simpler variation that doesn't have a specific name.

For probit classification, each potential solution is a set of weight values. **Figure 5** shows, in pseudocode, the variation of simplex optimization used in the demo program.

Simplex optimization, like all other ML optimization algorithms, has pros and cons. However, it's (relatively) simple to implement and usually—though not always—works well in practice.

The Demo Program

To create the demo program, I launched Visual Studio and selected the C# console application program template and named it ProbitClassification. The demo has no significant Microsoft .NET Framework version dependencies, so any relatively recent version of Visual Studio should work. After the template code loaded, in the Solution Explorer window I renamed file Program.cs to ProbitProgram.cs and Visual Studio automatically renamed class Program.

> Loosely speaking,
> a simplex is a triangle.

The beginning of the demo code is shown in **Figure 6**. The dummy data is hardcoded into the program. In a non-demo scenario, your data would be stored in a text file and you'd have to write a utility method to load the data into memory. Next, the source data is displayed using program-defined helper method ShowData:

Figure 5 **Pseudocode for the Simplex Optimization Used in the Demo Program**

```
randomly initialize best, worst other solutions
loop maxEpochs times
  create centroid from worst and other
  create expanded
  if expanded is better than worst, replace worst with expanded,
    continue loop
  create reflected
  if reflected  is better than worst, replace worst with reflected,
    continue loop
  create contracted
  if contracted  is better than worst, replace worst with contracted,
    continue loop
  create a random solution
  if  random solution is better than worst, replace worst,
    continue loop
  shrink worst and other toward best
end loop
return best solution found
```

Figure 6 **Beginning of the Demo Code**

```
using System;
namespace ProbitClassification
{
  class ProbitProgram
  {
    static void Main(string[] args)
    {
      Console.WriteLine("\nBegin Probit Binary Classification demo");
      Console.WriteLine("Goal is to predict death (0 = false, 1 = true)");
      double[][] data = new double[30][];
      data[0] = new double[] { 48, +1, 4.40, 0 };
      data[1] = new double[] { 60, -1, 7.89, 1 };
      // Etc.
      data[29] = new double[] { 68, -1, 8.38, 1 };
...
```

```
  Console.WriteLine("\nRaw data: \n");
  Console.WriteLine("       Age       Sex      Kidney   Died");
  Console.WriteLine("=====================================");
  ShowData(data, 5, 2, true);
```

Next, columns 0 and 2 of the source data are normalized:

```
  Console.WriteLine("Normalizing age and kidney data");
  int[] columns = new int[] { 0, 2 };
  double[][] means = Normalize(data, columns); // Normalize, save means & stdDevs
  Console.WriteLine("Done");
  Console.WriteLine("\nNormalized data: \n");
  ShowData(data, 5, 2, true);
```

The Normalize method saves and returns the means and standard deviations of all columns so that when new data is encountered, it can be normalized using the same parameters used to train the model. Next, the normalized data is split into a training set (80 percent) and test set (20 percent):

```
  Console.WriteLine("Creating train (80%) and test (20%) matrices");
  double[][] trainData;
  double[][] testData;
  MakeTrainTest(data, 0, out trainData, out testData);
  Console.WriteLine("Done");
  Console.WriteLine("\nNormalized training data: \n");
  ShowData(trainData, 3, 2, true);
```

You might want to parameterize method MakeTrainTest to accept the percentage of items to place in the training set. Next, a program-defined probit classifier object is instantiated:

```
  int numFeatures = 3; // Age, sex, kidney
  Console.WriteLine("Creating probit binary classifier");
  ProbitClassifier pc = new ProbitClassifier(numFeatures);
```

Figure 7 **The ProbitClassifier Class**

```
public class ProbitClassifier
{
  private int numFeatures; // Number of independent variables
  private double[] weights; // b0 = constant
  private Random rnd;

  public ProbitClassifier(int numFeatures) { . . }
  public double[] Train(double[][] trainData, int maxEpochs, int seed) { . . }
  private double[] Expanded(double[] centroid, double[] worst) { . . }
  private double[] Contracted(double[] centroid, double[] worst) { . . }
  private double[] RandomSolution() { . . }
  private double Error(double[][] trainData, double[] weights) { . . }
  public void SetWeights(double[] weights) { . . }
  public double[] GetWeights() { . . }
  public double ComputeOutput(double[] dataItem, double[] weights) { . . }
  private static double CumDensity(double z) { . . }
  public int ComputeDependent(double[] dataItem, double[] weights) { . . }
  public double Accuracy(double[][] trainData, double[] weights) { . . }

  private class Solution : IComparable<Solution>
  {
    // Defined here
  }
}
```

And then the probit classifier is trained, using simplex optimization to find values for the weights so that computed output values closely match the known output values:

```
  int maxEpochs = 100; // 100 gives a representative demo
  Console.WriteLine("Setting maxEpochs = " + maxEpochs);
  Console.WriteLine("Starting training");
  double[] bestWeights = pc.Train(trainData, maxEpochs, 0);
  Console.WriteLine("Training complete");
  Console.WriteLine("\nBest weights found:");
  ShowVector(bestWeights, 4, true);
```

The demo program concludes by computing the classification accuracy of the model on the training data and on the test data:

```
...
    double testAccuracy = pc.Accuracy(testData, bestWeights);
    Console.WriteLine("Prediction accuracy on test data = " +
testAccuracy.ToString("F4"));
    Console.WriteLine("\nEnd probit binary classification demo\n");
    Console.ReadLine();
  } // Main
```

The demo doesn't make a prediction for previously unseen data. Making a prediction could look like:

```
  // Slightly older, male, higher kidney score
  double[] unknownNormalized = new double[] { 0.25, -1.0, 0.50 };
  int died = pc.ComputeDependent(unknownNormalized, bestWeights);
  if (died == 0)
    Console.WriteLine("Predict survive");
  else if (died == 1)
    Console.WriteLine("Predict die");
```

This code assumes that the independent x variables—age, sex and kidney score—have been normalized using the means and standard deviations from the training data normalization process.

The ProbitClassifier Class

The overall structure of the ProbitClassifier class is presented in **Figure 7**. The ProbitClassifier definition contains a nested class named Solution. That sub-class derives from the IComparable interface so that an array of three Solution objects can be automatically sorted to give the best, other and worst solutions. Normally I don't like fancy coding techniques, but in this situation the benefit slightly outweighs the added complexity.

The ProbitClassifier has two output methods. Method ComputeOutput returns a value between 0.0 and 1.0 and is used during training to compute an error value. Method ComputeDependent is a wrapper around ComputeOutput and returns 0 if the output is less than or equal to 0.5, or 1 if the output is greater than 0.5. These return values are used to compute accuracy.

Wrapping Up

Probit classification is one of the oldest ML techniques. Because probit classification is so similar to logistic regression classification, common wisdom is to use either one technique or the other. Because LR is slightly easier to implement than probit, probit classification is used less often, and over time has become somewhat of a second-class ML citizen. However, probit classification is often very effective and can be a valuable addition to your ML toolkit. ■

Dr. James McCaffrey *works for Microsoft Research in Redmond, Wash. He has worked on several Microsoft products including Internet Explorer and Bing. Dr. McCaffrey can be reached at jammc@microsoft.com.*

Thanks to the following Microsoft Research technical experts for reviewing this article: *Nathan Brown and Kirk Olynyk.*

SQL Server LIVE!

TRAINING FOR DBAs AND IT PROS

ORLANDO NOV 17-21

ROYAL PACIFIC RESORT AT UNIVERSAL ORLANDO

LIVE! 360

TECH EVENTS WITH PERSPECTIVE

5 GREAT CONFERENCES. 1 GREAT PRICE.

- SQL Server LIVE! — TRAINING FOR DBAs AND IT PROS
- Visual Studio LIVE! — EXPERT SOLUTIONS FOR .NET DEVELOPERS
- SharePoint LIVE! — TRAINING FOR COLLABORATION
- Modern Apps LIVE! — MOBILE, CROSS-DEVICE & CLOUD DEVELOPMENT
- TECHMENTOR — IN-DEPTH TRAINING FOR IT PROS

live360events.com

CONQUER PLANET DATA

After 5 days of workshops, deep dives and breakout sessions, SQL Server Live! will leave you with the skills needed to Conquer Planet Data, whether you are a DBA, developer, IT Pro, or Analyst. With timely, relevant content delivered by recognized experts, SQL Server Live! provides comprehensive education and knowledge share on SQL Server database management, data warehouse/BI model design, Big Data analytics, performance tuning, troubleshooting and coding against SQL Server.

sqllive360.com

REGISTER BY OCTOBER 15 AND SAVE $300

Use promo code SQLOCT1

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

EVENT SPONSORS: Magenic, Microsoft

PLATINUM SPONSOR: esri

GOLD SPONSORS: New Relic., sitecore Own the experience

PRODUCED BY: 1105 MEDIA

RACHEL APPEL

Use SignalR to Build Modern Apps

With widespread access to broadband Internet and wireless devices, there's considerable demand for real-time apps. Popular sites such as Facebook and Twitter, multiplayer games and collaborative business apps have the best UX when they're live, real-time apps. Many other types of apps are great candidates for real-time experiences, such as stock and finance apps, auctions, sales dashboards, e-commerce and educational apps. Even Web sites and apps where live data isn't a necessity can benefit from real-time, full duplex communications with SignalR.

SignalR is a set of server and client libraries that facilitate simple, real-time, two-way communications between server and client.

What's SignalR and Why Should I Use It?

SignalR is a set of server and client libraries that facilitate simple, real-time, two-way communications between server and client. Not only can the client initiate contact with the server, as is the case in Web development, but the server can also contact the client. Those aren't just simple HTTP responses, either. These are actual method calls from server to client, like push technology. Clients can even contact other clients through the SignalR server-side component. All this is possible because SignalR creates a persistent connection between the server and client.

Everyone wants to create modern software—and it doesn't get any more modern than full duplex communications. There are several reasons to use SignalR. Its ease of use for writing Web sites and apps is one good reason. Another is if you need live communication in your software. In those instances, SignalR is the way to go. You could do it yourself using any number of techniques, such as WebSockets or AJAX polling. However, you'd have to rewrite all the groundwork the SignalR team has already done. This groundwork is fairly expansive and includes several key features:

- **Transport negotiation:** SignalR detects the best transport to get as close to real-time communications as possible. It uses WebSockets by default, as that's the fastest and most modern way to write real-time Web apps. Automatic transport management is behind the idea of real-time communication in SignalR. It negotiates a transport for each client as it connects.
- **SignalR server host:** You can choose between lightweight self-hosting anywhere, including non-Microsoft platforms, or hook SignalR into the IIS pipeline.
- **Client-side libraries:** Including JavaScript, the Microsoft .NET Framework and Windows Store libraries.
- **JavaScript proxy:** This provides a way to call methods in remote locations in JavaScript, while developing as if all code is running in the same process on the same machine.
- **Security:** SignalR hooks into existing ASP.NET security models, and supports many popular third-party security providers such as Microsoft Live, OpenAuth, Google, Facebook and Twitter.

Web developers traditionally write code according to the request/response model of HTTP. There's nothing inherently bad with this, but it lacks the primary benefit of SignalR—a persistent connection between server and client. In HTTP, you make a request, get a response and then you're done. In a real-time scenario, the pipeline stays open between the server and client. This lets you create richer and better UXes that feel alive and connected.

In SignalR, there are two layers of abstraction over the low-level transports—hubs and persistent connections. This article will cover just hubs for the sake of brevity. Hubs are the higher-level API that's the "incredibly simple" part of SignalR. Persistent connections take more time and effort to code, and SignalR uses them as the basis for hubs. You'll generally use hubs for most of your activities, unless you have good reasons to do otherwise.

Get Started with SignalR in Windows Apps

Like many other .NET libraries, SignalR comes as a NuGet package. You can install it with the NuGet Package Manager or the Package Manager Console. Both are features of Visual Studio 2012 and Visual Studio 2013. There are several different SignalR packages available from Microsoft, including:

- Microsoft ASP.NET SignalR: The base package that installs Core and Web components with the JavaScript client
- Microsoft ASP.NET SignalR Core Components: Hosting and core libraries

Code download available at github.com/rachelappel/VoteR.

- Microsoft ASP.NET SignalR System.Web: SignalR for ASP.NET
- Microsoft ASP.NET SignalR JavaScript Client: JavaScript client libraries for HTML apps
- Microsoft ASP.NET SignalR .NET Client: Client libraries for other Windows platform apps

When you install the Microsoft ASP.NET SignalR package in any of your ASP.NET projects (Web Forms or MVC), SignalR installs the dependencies in each of the packages listed except the .NET Client. The .NET Client is for Windows 8 and Windows Phone XAML apps, Windows Presentation Foundation (WPF), Windows Forms and Console apps. There are more SignalR packages in the NuGet Package Manager from both Microsoft and third parties. These cover just about everything including, self-hosting, scaling, dependency injection and MongoDB support.

After installation, there are no web.config settings to adjust. However, you must add a small snippet of startup code to indicate to ASP.NET that you'll be inserting SignalR into its pipeline:

```
[assembly: OwinStartup(typeof(VoteR.Startup))]
public partial class Startup
{
  public void Configuration(IAppBuilder app)
  {
    app.MapSignalR();
  }
}
```

You can add this Startup class to a .cs file in the App_Start folder. Some of the ASP.NET project templates already include a Startup class for ASP.NET Forms Authentication. If this is the case, just add the Configuration method to that class instead. Once you do this, you can move onto writing real-time code.

This Startup class is an Open Web Interface for .NET (OWIN) startup. That means it adheres to the new OWIN specifications. OWIN is a standard much like the standards governed by the Worldwide Web Consortium (W3C). Microsoft has implemented OWIN in a project called Katana. That's the engine behind SignalR that works in tandem with IIS, or as a self-host to facilitate two-way communications. As a SignalR developer using hubs, you won't need to know much more than that about OWIN or Katana. SignalR abstracts all that away so you can focus on solving your business problems with SignalR.

Figure 1 **The VoteHub Class Tallies Votes**

```
public class VoteHub : Hub
{
  private static List<Item> VoteItems = new List<Item>();
  private static VoteRContext db = new VoteRContext();

  public void Vote(int id)
  {
    var votes = AddVote(id);
    Clients.All.updateVoteResults(id, votes);
  }
  private static Item AddVote(int id) {
    var voteItem = VoteItems.Find(v => v.Id == id);
    if (voteItem != null)
    {
      voteItem.Votes++;
      return voteItem;
    }
    else
    {
      var item = db.Items.Find(id);
      item.Votes++;
      VoteItems.Add(item);
      return item;
    }
  }
  public override Task OnConnected()
  {
    Clients.Caller.joinVoting(VoteItems.ToList());
    return base.OnConnected();
  }
}
```

Figure 2 **Various Ways to Communicate from the Server to the Client**

```
// Call method on all clients
Clients.All.clientSideMethod(args, args, ...);
// Call method on specific client
Clients.Client(Context.ConnectionId). clientSideMethod(args, args, ...);
// Call a method on a list of specific connections
Clients.Clients(ConnectionId1, ConnectionId1, ...).clientSideMethod(args, args, ...);
// Call method on calling connection
Clients.Caller.clientSideMethod(args, args, ...);
// Call method on all clients but the caller
Clients.Others. clientSideMethod(args, args, ...);
// Call method on all in the group except a few select connections
Clients.AllExcept(connectionId1, connectionId2).clientSideMethod(args, args, ...);
// Call method on groups of connections
Clients.Group(groupName).clientSideMethod(args, args, ...);
// Call method on connected clients in a specified group
Clients.Groups(GroupIds).clientSideMethod(args, args, ...);
// Call method on other connected clients in a specified group
Clients.OthersInGroup(groupName).clientSideMethod(args, args, ...);
```

The Server Side of SignalR Hubs

Hubs are the communications core of SignalR. They receive incoming requests and push messages to clients, either from the hub itself or on behalf of another client. You can consider a hub in SignalR to be like a hub and spoke of a wheel.

The hub is just a gatekeeper for the messaging. While they're at the center of the action, hubs are just classes that inherit from the Microsoft.AspNet.SignalR.Hub class. The Hub class, in turn, implements the IHub interface of the Microsoft.AspNet.SignalR.Hubs namespace. The IHub interface defines three events: OnConnected, OnDisconnected and OnReconnected. It also defines three properties: Clients, Context and Groups. These are common tasks or information related to each real-time connection to the hub.

The client calls public methods on the hub, which means the code looks like a Web Service call. However, SignalR hubs can initiate contact with clients that have registered with them. You don't normally program with this kind of behavior in mind, as you'd normally use the traditional request/response model.

This can happen because of the Clients property that represents a collection of all the connected clients. Through the Clients property, you can access a single client or multiple clients regardless of their platform. For example, an iOS client can send a message to a Windows client through the hub as the hub communicates with the Windows client on the iOS client's behalf and vice versa.

To see the hub in action, I'll examine a sample app called VoteR that displays a series of items and lets users vote for their favorite. At the app's core, there's a VoteHub class. This is the hub that tallies the votes users have cast for each item. It then notifies clients of the updated numbers. **Figure 1** shows an example of the VoteHub class.

The two methods to investigate in **Figure 1** are the Vote and AddVote methods. The Vote method is what the clients call (covered in the next section). The app then calls the private AddVote method,

which does the actual vote counting. It does so by checking to see if items are already in the VoteItems list. If they are, it will update them. If not, the first time a user votes for them, AddVote will add that item. A static List<Vote> is an easy way to store simple global information such as this without a database.

The Vote method in **Figure 1** contains an interesting line of code after its call to AddVote:

```
Clients.All.updateVoteResults(id, votes);
```

In SignalR, you'll use the Clients property to access and call code on the client. From the Clients property, you can target the clients you want to access. Sometimes it's all clients, such as when a new vote happens. Sometimes it's only one client, such as when a user first connects. Using naming conventions lets SignalR match the server call with the code on the client to run. This is the first time in the history of ASP.NET that you can use dynamic properties to call code on the client.

This is the first time in the history of ASP.NET that you can use dynamic properties to call code on the client.

As you might imagine, because the VoteHub needs to track votes, it makes sense that there's an event like OnConnected. The OnConnected event lets you capture new, incoming connections. A likely scenario is to trap the connection's id through the ConnectionId property of the Context object.

In the case of the VoteR app, in **Figure 1** it's cycling through the three items in the VoteHub List<Item> object and notifying the client how many votes each item has via the Clients.caller property. This way, newly connected clients immediately have the total number of votes for each item upon joining the party.

There are many other ways to communicate between server and client. The Clients property of the Hub class exposes the many various ways we can access client code, as outlined in **Figure 2**.

In **Figure 2**, the clientSideMethod is defined on the client the server calls. In the next section, you'll learn how to define these methods on the client. As you can see, the dynamic nature of the Clients property lets you write code for a wide range of server-to-client-communication scenarios.

Figure 3 **JavaScript Client Code**

```
$(function () {
  var voteHub = $.connection.voteHub;
  $.connection.hub.start().done(function () {
    $("button").click(function () {
      voteHub.server.vote(this.id);
      });
  });
  voteHub.client.updateVoteResults = function (id, vote) {
    // Update UI to show each item and how many votes it has
    }
  voteHub.client.joinVoting = function (votes) {
    // Cycle through votes to display current information
    // about each item to newcomer
  }
});
```

The Client Side of SignalR: JavaScript and .NET Clients

You can build real-time apps on any platform with SignalR. Out of the box, you can use the SignalR JavaScript client for all things Web and HTML client apps, including WinJS apps. Plain HTML and JavaScript are widely supported languages. For the .NET folks, Microsoft has released a .NET client for both Windows and desktop apps. Like the core components, you install either the JavaScript or .NET Client from NuGet, depending on the type of your project. Because JavaScript is just JavaScript, you can download the scripts from github.com/SignalR/SignalR, and add <script> tags to your page, as opposed to referencing .dll files, as shown here:

```
<script src="~/Scripts/jquery-1.10.2.js"></script>
<script src="~/Scripts/jquery.signalR-2.0.3.js"></script>
<script src="~/signalr/hubs"></script>
```

The order of the script references is important. You must load jQuery first because SignalR depends on it. Next in line is the SignalR client. The last one is the SignalR proxy. SignalR dynamically generates a proxy at run time and drops it off at /signalr/hubs. This proxy is what lets you write code on both the client and server, yet have it behave as if it's all in the same location.

The VoteR app's client script defines methods to receive calls from the server, as well as ordinary methods and event wire-ups. In **Figure 3**, the very first line of code traps a variable called voteHub. This is a direct line to an instance of the VoteHub class. SignalR creates an instance of the hub for each client that connects. The client starts the connection with a call to $.connection.hub.start that returns a promise. This means the code within won't run until it's complete. In this case, it's a call to the Vote method on the server inside the voting button's click event. As you can see, it passes the item id for which the user is voting to the server. Then the server does the work outlined in **Figure 1**.

You can build real-time apps on any platform with SignalR. Out of the box, you can use the SignalR JavaScript client for all things Web and HTML client apps, including WinJS apps.

At this point, you might think there are typos in **Figure 3**. That's because naming the VoteHub class and Vote method is inconsistent between server and client. That's not a typo, but rather a SignalR convention. In JavaScript clients, calls to hub.server.methodName go in camelCase by default. It's easy enough to change this behavior by attaching the HubName attribute to the Hub class with the exact capitalization you want. The HubName attribute looks like this: HubName("VoteHub").

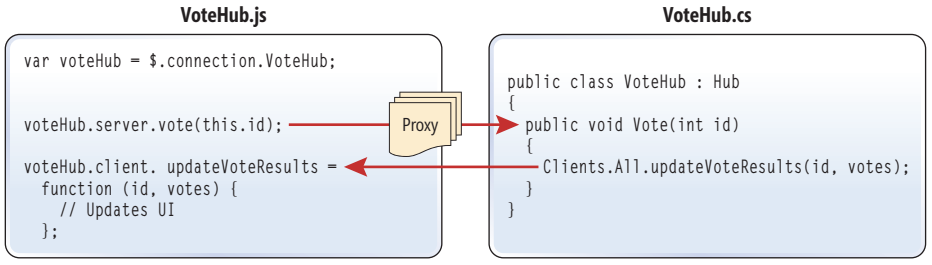


Figure 4 **The Relationship Between Hub and Client Method Calls**

The two most interesting pieces of code in **Figure 3** are the voteHub.client.updateVoteResults and voteHub.client.joinVoting blocks. As their signatures indicate, both methods are members of the Client property of the VoteHub on the server. Reflecting back on **Figure 1**, the client-side voteHub.client.updateVoteResults method from **Figure 3** aligns with the Clients.All.updateVoteResults(id, votes) call from **Figure 1**. **Figure 4** shows the relationship between the server and client code.

Now, it's time to examine the .NET client. **Figure 5** shows some code that makes a connection from a Windows Store XAML app using C#. This could just as easily be a Windows Phone app, as the code is identical. It starts by creating a connection to the SignalR pipeline. Then it goes on to create the proxy.

What you see in **Figure 5** that you won't normally see in the JavaScript client is the HTTP path to the SignalR pipeline passed into the HubConnection constructor. Unlike the JavaScript client, it's slightly less automatic. You must instantiate a HubConnection and call CreateHubProxy to create the proxy.

Notice in **Figure 5** there's an assignment that obtains a synchronization context. You wrap the client code the server calls with this context object. Then you call its Post method and pass a delegate. Delegates in C# are the same as inline anonymous functions in JavaScript.

After starting the hub's connection, you can call its proxy's Invoke method to vote on an item. Using the await keyword causes it to perform those actions asynchronously. You can find the complete source for the VoteR demo application at: github.com/rachelappel/VoteR.

SignalR Deployment: Server and Client

Because SignalR is ASP.NET, you must deploy to an environment with the .NET Framework 4.5 or later, unless you self-host. In an ASP.NET project, SignalR is just a set of libraries. It goes along with the other libraries when it's time to deploy.

If you think the rest of SignalR is easy, wait until you try to deploy to Microsoft Azure. Using Azure makes the deployment process especially stress-free. This also deploys both the SignalR server component and at least one HTML client to a Web server. Of course, you must publish any Windows Store or Windows Phone apps to the app store, and desktop apps to desktops through their appropriate channels.

Figure 5 **Windows Store C# Client Code to Vote in VoteR**

```
async private Task startConnection()
{
  var hubConnection = new HubConnection("http://localhost:25024/signalr");
  IHubProxy hubProxy = hubConnection.CreateHubProxy("VoteHub");
  var context = SynchronizationContext.Current;

  hubProxy.On<string, string>("updateVoteResults", (id, votes) =>
    context.Post(delegate {
  // Update UI
); }, null));

  await hubConnection.Start();
  await hubProxy.Invoke("vote", "rachel", Convert.ToDecimal(itemId.Text));
}
```

Those with any ASP.NET projects can choose Publish from the Visual Studio Build menu to start the deployment to Azure. If you're using Visual Studio 2013, you can just follow the prompts. You'll only need to enter your credentials and choose the database and Web site name.

> If you think the rest of SignalR is easy, wait until you try to deploy to Microsoft Azure.

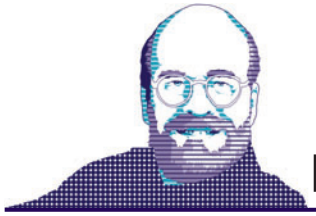
In Visual Studio 2012, it's a similar set of prompts. During deployment, you can choose to create a new Azure Web site or select an existing one. If it's an existing site, log in to the Azure Portal, navigate to the Configuration tab, and find and turn on WebSockets. You must do this with a new Web site, as well, but Visual Studio will create and launch the site first, which will cause an error. Again, just log in and turn on WebSockets. That's the important thing. It's a good idea to stop and start any Web sites after a configuration change.

Wrapping Up

SignalR really is simple, real-time programming. While it's an ASP.NET product, it's cross-platform in that you can write Windows Store, iOS and Android apps with an ASP.NET server component. You can also self-host on non-Microsoft OSes. This makes SignalR flexible, as well as simple and efficient. Another great thing about SignalR is that you don't even need to have real-time functionality as a requirement. Use it going forward and join the many who have already adopted the real-time programming paradigm. ■

Rachel Appel *is a consultant, author, mentor and former Microsoft employee with more than 20 years of experience in the IT industry. She speaks at top industry conferences such as Visual Studio Live!, DevConnections, MIX and more. Her expertise lies within developing solutions that align business and technology focusing on the Microsoft dev stack and open Web. For more about Appel, visit her Web site at rachelappel.com.*

Thanks to the following Microsoft technical expert for reviewing this article: *Frank La Vigne*



CHARLES PETZOLD

Pixel Shaders and the Reflection of Light

If you could see photons … well, you *can* see photons, or at least some of them. Photons are the particles that make up electromagnetic radiation, and the eye is sensitive to photons with wavelengths within the range of visible light.

But you can't see photons as they fly about all over the place. That would surely be interesting. Sometimes photons go right through objects; sometimes they're absorbed; sometimes they're reflected; but often it's a combination of all of these effects. Some of the photons that bounce off objects eventually reach your eyes, giving each object its particular color and texture.

For extremely high-quality 3D graphics, a technique called ray tracing can actually plot out a simulation of the paths of these myriad photons to mimic the effects of reflection and shadows. But much simpler techniques are available for more conventional needs. This is often the case when using Direct3D—or, in my case, writing custom effects in Direct2D that make use of 3D.

Reusing the Effect

As you've seen in previous installments of this column, a Direct2D effect is basically a wrapper for code that runs on the GPU. Such code is known as a shader, and the most important are the vertex shader and the pixel shader. The code in each of these shaders is called at the video refresh rate of the display. The vertex shader is called for each of the three vertices in each triangle that make up the graphical objects displayed by the effect, while the pixel shader is called for every pixel within these triangles.

Obviously, the pixel shader is called much more frequently than the vertex shader, so it makes sense to keep as much processing as possible in the vertex shader rather than the pixel shader. This isn't always possible, however, and when using these shaders to simulate the reflection of light, it's usually the balance and interaction between these two shaders that governs the sophistication and flexibility of the shading.

In the August issue of this magazine, I presented a Direct2D effect called RotatingTriangleEffect that constructed a vertex buffer consisting of points and colors, and allowed applying standard model and camera transforms to the vertices. I used this effect for rotating three triangles. That's not a lot of data. Three triangles involve just a total of nine vertices, and I mentioned at the time that the same effect could be used for a much larger vertex buffer.

Code download available at msdn.microsoft.com/magazine/msdnmag1014.

Let's try it out: The downloadable program (msdn.microsoft.com/magazine/msdnmag1014) for this column is called ShadedCircularText, and it uses RotatingTriangleEffect without a single change.

The ShadedCircularText program returns to the problem I began exploring earlier this year of displaying tessellated 2D text in three dimensions. The constructor of the ShadedCircularTextRenderer class loads in a font file, creates a font face from it, and then calls GetGlyphRunOutline to obtain a path geometry of the character outlines. This path geometry is then tessellated using a class I created called InterrogableTessellationSink that accumulates the actual triangles.

After registering RotatingTriangleEffect, ShadedCircularTextRenderer creates an ID2D1Effect object based on this effect. It then converts the triangles of the tessellated text into vertices on the surface of a sphere, basically wrapping the text around the equator and bending it toward the poles. The color of each vertex is based on a hue derived from the X coordinate of the original text geometry. This creates a rainbow-like effect, and **Figure 1** shows the result.

As you can see, a little menu adorns the top. The program actually incorporates three additional Direct2D effects that implement more traditional shading models. All of them use the same points, the same transforms, and the same animations, so you can switch between them to see the difference. The differences involve only the color shading of the triangles.

The bottom-right corner has a display of performance in frames-per-second, but you'll discover that nothing in this program causes that to drop much below 60 except if something else is going on.

Gouraud Shading

As photons are flying all around us, they often bounce off nitrogen and oxygen molecules in the air. Even on an overcast day with no

Figure 1 **A 3D Text Rainbow from ShadedCircularText**

Figure 2 **The Vertex Shader for Gouraud Shading**

```
// Per-vertex data input to the vertex shader
struct VertexShaderInput
{
  float3 position : MESH_POSITION;
  float3 normal : NORMAL;
  float3 color : COLOR0;
  float3 backColor : COLOR1;
};

// Per-vertex data output from the vertex shader
struct VertexShaderOutput
{
  float4 clipSpaceOutput : SV_POSITION;
  float4 sceneSpaceOutput : SCENE_POSITION;
  float3 color : COLOR0;
};

// Constant buffer provided by effect.
cbuffer VertexShaderConstantBuffer : register(b1)
{
  float4x4 modelMatrix;
  float4x4 viewMatrix;
  float4x4 projectionMatrix;
  float4 ambientLight;
  float4 directionalLight;
  float4 lightDirection;
};

// Called for each vertex.
VertexShaderOutput main(VertexShaderInput input)
{
  // Output structure
  VertexShaderOutput output;

  // Get the input vertex, and include a W coordinate
  float4 pos = float4(input.position.xyz, 1.0f);

  // Pass through the resultant scene space output value
  //  (not necessary -- can be removed from both shaders)
  output.sceneSpaceOutput = pos;

  // Apply transforms to that vertex
  pos = mul(pos, modelMatrix);
  pos = mul(pos, viewMatrix);
  pos = mul(pos, projectionMatrix);

  // The result is clip space output
  output.clipSpaceOutput = pos;

  // Apply model transform to normal
  float4 normal = float4(input.normal, 0);
  normal = mul(normal, modelMatrix);

  // Find angle between light and normal
  float3 lightDir = normalize(lightDirection.xyz);
  float cosine = -dot(normal.xyz, lightDir);
  cosine = max(cosine, 0);

  // Apply view transform to normal
  normal = mul(normal, viewMatrix);

  // Check if normal pointing at viewer
  if (normal.z > 0)
  {
    output.color = (ambientLight.xyz + cosine *
                    directionalLight.xyz) * input.color;
  }
  else
  {
    output.color = input.backColor;
  }

  return output;
}
```

direct sunlight, there's still a considerable amount of ambient light. Ambient light tends to illuminate objects in a very uniform manner.

Perhaps you have an object that's a greenish blue, with an RGB value of (0, 0.5, 1.0). If the ambient light is one-quarter of full intensity white, you can assign an RGB value to the light of (0.25, 0.25, 0.25). The perceived color of this object is the product of the red, green, and blue components of these numbers, or (0, 0.125, 0.25). It's still a greenish blue, but much darker.

But simple 3D scenes don't live by ambient light alone. In real life, objects normally have a lot of color variation on their surfaces, so even if they're uniformly illuminated, the objects still have visible textures. But in a simple 3D scene, a greenish-blue object illuminated only by ambient light will merely look like an undifferentiated slab of uniform color.

For this reason, simple 3D scenes benefit enormously from some directional light. It's simplest to assume that this light comes from a far distance (like the sun), so the direction of the light is just a single vector that applies to the entire scene. If there's only one light source, generally it's assumed to come from over the viewer's left shoulder, so perhaps the vector is (1, -1, -1) in a right-hand coordinate system. This directional light also has a color, perhaps (0.75, 0.75, 0.75), so together with the ambient light of (0.25, 0.25, 0.25), maximum illumination is sometimes achieved.

The amount of directional light a surface reflects depends on the angle the light makes with the surface. (This is a concept explored in my May 2014 DirectX Factor column.) The maximum reflection occurs when the directional light is perpendicular to the surface, and reflected light decreases to zero when the light is tangent to the surface or coming from somewhere behind the surface.

The Lambert Cosine Law—named after German mathematician and physicist Johann Heinrich Lambert (1728 – 1777)—says that the fraction of light reflected from a surface is the negative cosine of the angle between the direction of the light, and the direction of a vector perpendicular to the surface, which is called a surface normal. If these two vectors are normalized—that is, if they have a magnitude of 1—this cosine of the angle between the two vectors is the same as the dot product of the vectors.

For example, if the light strikes a particular surface at a 45-degree angle, the cosine is approximately 0.7, so multiply that by the directional light color of (0.75, 0.75, 0.75) and the color of the object (0, 0.5, 1.0) to derive the color of the object from directional light of (0, 0.26, 0.53). Add that to the color from ambient light.

Figure 3 **The Pixel Shader for Gouraud Shading**

```
// Per-pixel data input to the pixel shader
struct PixelShaderInput
{
  float4 clipSpaceOutput : SV_POSITION;
  float4 sceneSpaceOutput : SCENE_POSITION;
  float3 color : COLOR0;
};

// Called for each pixel
float4 main(PixelShaderInput input) : SV_TARGET
{
  // Simply return color with opacity of 1
  return float4(input.color, 1);
}
```

However, keep in mind that curved objects in 3D scenes aren't actually curved. Everything in the scene consists of flat triangles. If the illumination of each triangle is based on a surface normal perpendicular to the triangle itself, each triangle will have a different uniform color. This is fine for the Platonic solids such as those displayed in my May 2014 column, but not so good for curved surfaces. For curved surfaces, you want the colors of the triangles to blend with each other.

This means it's necessary for each triangle to have a graduated color rather than a uniform color. The color from directional light can't be based on a single surface normal for the triangle. Instead, each vertex of the triangle should have a different color based on the surface normal at that vertex. These vertex colors can then be interpolated over all the pixels of the triangle. Adjacent triangles then blend in with each other to resemble a curved surface.

This type of shading was invented by French computer scientist Henri Gouraud (b. 1944) in a paper published in 1971, and is therefore known as Gouraud shading.

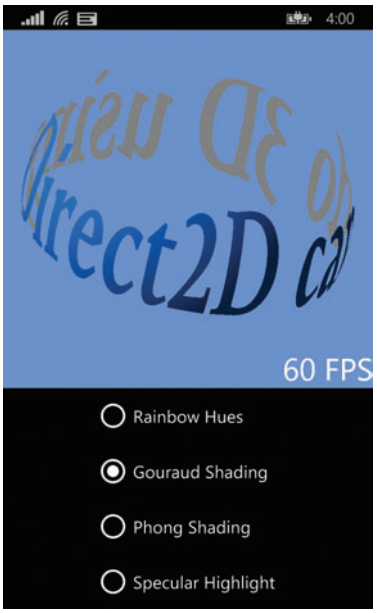


Figure 4 **The Display of the Gouraud Shading Model**

Gouraud shading is the second option implemented in the ShadedCircularText program. The effect itself is called GouraudShadingEffect, and it requires a vertex buffer with somewhat more data:

```
struct PositionNormalColorVertex
{
  DirectX::XMFLOAT3 position;
  DirectX::XMFLOAT3 normal;
  DirectX::XMFLOAT3 color;
  DirectX::XMFLOAT3 backColor;
};
```

Interestingly, because the text is effectively being wrapped around a sphere centered at the point (0, 0, 0), the surface normal at each vertex is the same as the position, but normalized to have a magnitude of 1. The effect allows unique colors for every vertex, but in this program every vertex gets the same color, which is (0, 0.5, 1) and the same backColor of (0.5, 0.5, 0.5), which is the color to be used if the back of a surface faces the viewer.

The GouraudShadingEffect also requires more effect properties. It must be possible to set the ambient light color, directional light color, and the vector direction of the directional light. The GouraudShadingEffect transfers all these values to a larger constant buffer for the vertex shader. The vertex shader itself is shown in **Figure 2**.

The pixel shader is the same as for the RotatingTriangleEffect, and is shown in **Figure 3**. The interpolation of the vertex colors over the entire triangle occurs behind the scenes between the vertex shader and the pixel shader, so the pixel shader simply passes the color on to be displayed.

The result is shown in **Figure 4**, this time on Windows Phone 8.1 rather than Windows 8.1. The ShadedCircularText solution was created in Visual Studio with the new Universal App template and can be compiled for either platform. All code is shared between the two platforms except for the App and DirectXPage classes. The difference in the layouts of the two programs suggests why having different page definitions is often a good idea, even if the functionality of the program is fundamentally the same.

Figure 5 **The Vertex Shader for the Phong Shading Model**

```
// Per-vertex data input to the vertex shader
struct VertexShaderInput
{
  float3 position : MESH_POSITION;
  float3 normal : NORMAL;
  float3 color : COLOR0;
  float3 backColor : COLOR1;
};

// Per-vertex data output from the vertex shader
struct VertexShaderOutput
{
  float4 clipSpaceOutput : SV_POSITION;
  float4 sceneSpaceOutput : SCENE_POSITION;
  float3 normalModel : NORMAL0;
  float3 normalView : NORMAL1;
  float3 color : COLOR0;
  float3 backColor : COLOR1;
};

// Constant buffer provided by effect
cbuffer VertexShaderConstantBuffer : register(b1)
{
  float4x4 modelMatrix;
  float4x4 viewMatrix;
  float4x4 projectionMatrix;
};

// Called for each vertex
VertexShaderOutput main(VertexShaderInput input)
{
  // Output structure
  VertexShaderOutput output;

  // Get the input vertex, and include a W coordinate
  float4 pos = float4(input.position.xyz, 1.0f);

  // Pass through the resultant scene space output value
  // (not necessary – can be removed from both shaders)
  output.sceneSpaceOutput = pos;

  // Apply transforms to that vertex
  pos = mul(pos, modelMatrix);
  pos = mul(pos, viewMatrix);
  pos = mul(pos, projectionMatrix);

  // The result is clip space output
  output.clipSpaceOutput = pos;

  // Apply model transform to normal
  float4 normal = float4(input.normal, 0);
  normal = mul(normal, modelMatrix);
  output.normalModel = normal.xyz;

  // Apply view transform to normal
  normal = mul(normal, viewMatrix);
  output.normalView = normal.xyz;

  // Transfer colors
  output.color = input.color;
  output.backColor = input.backColor;

  return output;
}
```

As you can see, the figure is lighter in its upper-left area, clearly showing the effect of directional light and aiding in the illusion of a rounded appearance to the surface.

The Phong Improvements

Gouraud shading is a time-honored technique, but it has a fundamental flaw: In Gouraud shading, the amount of directional light reflected in the center of a triangle is an interpolated value of the light reflected at the vertices. The light reflected at the vertices is based on the cosine of the angle between the light direction and the surface normals at those vertices.

But the light reflected in the center of the triangle should really be based on the surface normal at that point. In other words, the colors shouldn't be interpolated over the triangle; instead, the surface normals should be interpolated over the triangle's surface, and the reflected light calculated for each pixel based on that normal.

Enter Vietnamese-born computer scientist Pui Tuong Phong (1942-1975), who died of leukemia at the age of 32. In his 1973 doctoral dissertation, Phong described a somewhat different shading algorithm. Rather than interpolate vertex colors over the triangle, the vertex normals are interpolated over the triangle, and then reflected light is calculated from those.

In a practical sense, Phong shading requires the calculation of reflected light to be moved from the vertex shader to the pixel shader, along with the section of the constant buffer devoted to that job. This increases the amount of per-pixel processing immensely, but, fortunately, it's being done on the GPU where you hope it won't seem to make much of a difference.

The vertex shader for the Phong shading model is shown in **Figure 5**. Some of the input data—such as the color and back color—are simply passed on the pixel shader. But it's still useful to apply all the transforms here. The world transform and both camera transforms must be applied to the positions, while two normals are also calculated—one with only the model transform for the reflected light, and another with the view transform to determine whether a surface faces toward or away from the viewer.

As the output from the vertex shader becomes input to the pixel shader, these normals are interpolated over the triangle's surface. The pixel shader can then finish up the job by calculating the reflected light, as shown in **Figure 6**.

However, I'm not going to show you a screenshot of the result. It's pretty much visually identical to the Gouraud shading. Gouraud shading really is a good approximation.

Specular Highlights

The real importance of Phong shading is that it makes possible other features that rely upon a more accurate surface normal.

So far in this article, you've seen shading that's appropriate for diffuse surfaces. These are surfaces that are rather rough and dull and that tend to scatter light reflected off their surfaces.

Figure 6 **The Pixel Shader for the Phong Shading Model**

```
// Per-pixel data input to the pixel shader
struct PixelShaderInput
{
  float4 clipSpaceOutput : SV_POSITION;
  float4 sceneSpaceOutput : SCENE_POSITION;
  float3 normalModel : NORMAL0;
  float3 normalView : NORMAL1;
  float3 color : COLOR0;
  float3 backColor : COLOR1;
};

// Constant buffer provided by effect
cbuffer PixelShaderConstantBuffer : register(b0)
{
  float4 ambientLight;
  float4 directionalLight;
  float4 lightDirection;
};

// Called for each pixel
float4 main(PixelShaderInput input) : SV_TARGET
{
  // Find angle between light and normal
  float3 lightDir = normalize(lightDirection.xyz);
  float cosine = -dot(input.normalModel, lightDir);
  cosine = max(cosine, 0);
  float3 color;

  // Check if normal pointing at viewer
  if (input.normalView.z > 0)
  {
    color = (ambientLight.xyz + cosine *
      directionalLight.xyz) * input.color;
  }
  else
  {
    color = input.backColor;
  }

  // Return color with opacity of 1
  return float4(color, 1);
}
```

A surface that's somewhat glossy reflects light a little differently. If a surface is tilted just so, directional light could bounce off and go straight to the viewer's eye. This is usually perceived as bright white light, and it's known as a specular highlight. You can see the rather exaggerated effect in **Figure 7**. If the figure had sharper curves, the white light would be more localized.

Getting this effect seems at first as if it might be computationally complex, but it's only a few lines of code in the pixel shader. This particular technique was developed by NASA graphics maven Jim Blinn (b. 1949).

We first need a vector indicating the direction that the viewer of the 3D scene is looking. This is very easy because the view camera transform has adjusted all coordinates so the viewer is looking straight down the Z axis:

```
float3 viewVector = float3(0, 0, -1);
```

Next, calculate a vector that's halfway between that view vector and the light direction:

```
float3 halfway = -normalize(viewVector + lightDirection.xyz);
```

Notice the negative sign. This makes the vector point in the opposite direction—midway between the source of the light and the viewer.

If a particular triangle contains a surface normal that corresponds exactly with this halfway vector, it means that light is bouncing off the surface directly into the viewer's eye. This results in maximum specular highlighting.

Lesser highlighting results from non-zero angles between the halfway vector and the surface normal. This is another application

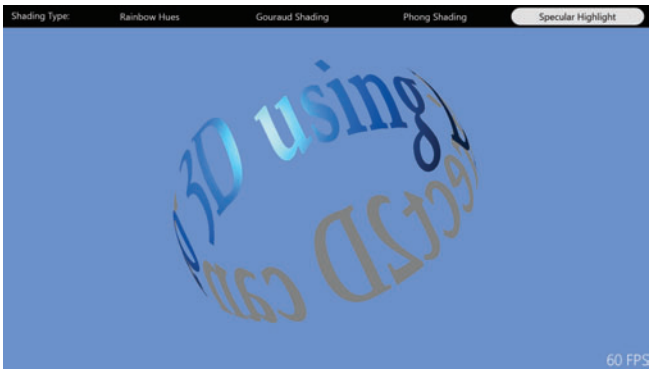


Figure 7 **The Specular Highlight Display**

for the cosine between the two vectors, which is the same as the dot product if the two vectors are normalized:

```
float dotProduct = max(0.0f, dot(input.normalView, halfway));
```

This value of dotProduct ranges from 1 for maximum specular highlighting when the angle between the two vectors is 0, to 0 for no specular highlighting, which occurs when the two vectors are perpendicular.

However, specular highlighting shouldn't be visible for all angles between 0 and 90 degrees. It should be localized. It should exist only for very small angles between those two vectors. You need a function that won't affect a dot product of 1, but will cause values less than 1 to become much lower. This is the pow function:

```
float specularLuminance = pow(dotProduct, 20);
```

This pow function takes the dot product to the 20th power. If the dot product is 1, the pow function returns 1. If the dot product is 0.7 (which results from an angle of 45 degrees between the two vectors), then the pow function returns 0.0008, which is effectively 0 as far as lighting goes. Use higher exponent values to make the effect even more localized.

Now all that's necessary is to multiply this factor by the directional light color and add it to the color already calculated from ambient light and directional light:

```
color += specularLuminance * directionalLight.xyz;
```

That creates a splash of white light as the animation turns the figure.

Farewell

And with that, the DirectX Factor column comes to a close. This plunge into DirectX has been one of the most challenging jobs of my career, but consequently also one of the most rewarding, and I hope to have an opportunity someday to return to this powerful technology. ■

Charles Petzold *is a longtime contributor to* MSDN Magazine *and the author of "Programming Windows, 6th Edition" (Microsoft Press, 2013), a book about writing applications for Windows 8. His Web site is charlespetzold.com.*

Thanks to the following Microsoft technical expert for reviewing this article: *Doug Erickson*

This issue marks Charles Petzold's last as a regular columnist in *MSDN Magazine*. Charles is leaving to join the team at Xamarin, a leading provider of cross-platform tools leveraging the Microsoft .NET Framework. Charles has been associated with *MSDN Magazine* for decades and has authored numerous regular columns, including Foundations, UI Frontiers and DirectX Factor. We wish him well on his new endeavors.

MSDN Magazine Online

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

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

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

msdn magazine

Raising Higher Education

I haven't been this excited about my fall teaching schedule in more than a decade. I'm rolling out a new class at Harvard University Extension School called CSCI-E34, User Experience Engineering. It's an academic adaptation of the industrial UX material that I teach through IDesign. Its subtitle, which is also the title of my forthcoming book, is "The Joy of UX."

I have 77 students registered as I write these words (late August), with a week left in the registration period. That figure really shows the emerging recognition of the importance of UX, which I've been shouting about since I started writing this column nearly five years ago. Last year I declared the 2010s the Decade of UX (msdn.microsoft.com/magazine/dn342880). It's starting to roll.

> UX isn't about programming. It's about figuring out what ought to be programmed. That's an entirely different problem, to which you cannot easily Google a solution, as you can to, say, implementing a color gradient.

My students will work throughout the term on a single UX project of their own choosing, developing their materials as I teach each topic. The end result will be a complete UX design package: personas, stories, preliminary layout and storyboards, UX testing plan, telemetry plan, security analysis, and prototype implementation. In addition to gaining hands-on experience with each stage of the design process, they finish my class with a killer package to show prospective employers.

Interestingly, about 25 percent of my students are female, at least judging by their names on my class roster. (I don't know many men named "Lisa," although silverbacks like me might remember Johnny Cash's classic song, "A Boy Named Sue" [bit.ly/1un5XFV].) That's the highest percentage of female students I've ever had in a class. As a father of two daughters and no sons, I think that's great. The industry is finally paying attention to what my daughter Annabelle wrote in this column last month (that didn't take long, did it?).

My class doesn't teach graphical design, otherwise known as decoration. I wrote about that here two years ago (msdn.microsoft.com/magazine/hh394140). Craigslist single-handedly brought down the entire newspaper industry with killer content and ultra-simple usage, and no graphical design whatsoever. (I am, however, providing a very good graphical designer as a guest speaker to rebut this contention.)

My class doesn't teach implementation, either. UX isn't about programming. It's about figuring out what ought to be programmed. That's an entirely different problem, to which you cannot easily Google a solution, as you can to, say, implementing a color gradient.

Because the world needs good UX people quickly, I've adapted an idea from the Army. Wounded soldiers need *immediate* medical attention (the "golden hour" of trauma medicine), so the Army provides each platoon with a medic. The medic isn't a fully qualified doctor, although the soldiers customarily address him as "Doc." The medic is trained in the Army's protocols for stabilizing wounded soldiers—opening airways, stopping severe bleeding, starting IVs and so on.

Similarly, UX questions that arise in development projects require quick answers. You can't have untrained developers doing it on their own ("Code 0x80040005 – Unknown error").

My class aims to create UX medics, so that every development team can afford one. The UX medic will know the basic concepts of UX design and their most common applications—for example, knowing that data is the key to most UX questions, and how to start obtaining it. She will know how to generate a user persona quickly and accurately, to help the development team grasp the slippery concept of "the user." She will know how to do a usability test quickly and cheaply so it doesn't hold up the project, or get skipped to keep it from holding up the project. Above all, she will know how to iterate the UX, starting early and continuing throughout the project.

If you have a UX guru or a team of them (as the Army has actual surgeons), their time is in very tight supply. They will benefit greatly from having trained UX medics on the project teams to handle the small stuff immediately, and to package up the hardest stuff for them to handle.

As the Army provides its medics with protocols, so I've prepared directions and templates for my students, the forthcoming UX medics. With characteristic modesty, I call this assemblage the Plattski® Protocol™. Seventy-seven students (and counting) are about to start learning it.

I'm really looking forward to spreading the Joy of UX. ■

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.*

.NET Tools for the Professional Developer

Data Visualization, User Interface, Reporting and Spreadsheet Controls Experts

ActiveReports 8

Create sophisticated, fast and powerful reports

Studio Enterprise

Hundreds of UI controls for all .NET platforms including grids, charts, reports and schedulers

Spread Studio

Flexible and familiar spreadsheet architecture, advanced charting and powerful formula library

Download your free trials at
ComponentOne.com

EXPECT MORE. GET MORE.

ComponentOne®
a division of GrapeCity®

Aspose.Cells

Work with spreadsheets and data without depending on Microsoft Excel

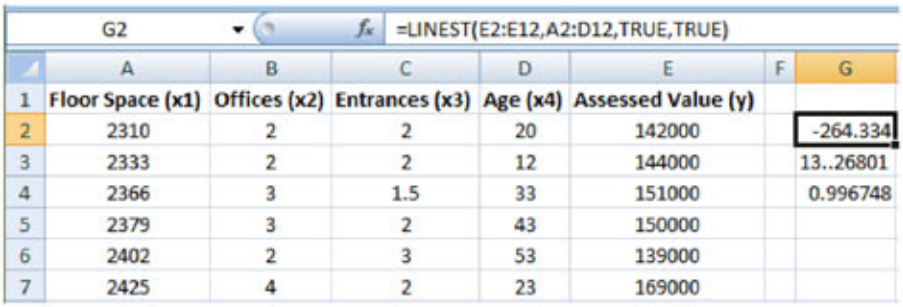
- Solution for spreadsheet creation, manipulation and conversion.
- Import and export data.

ASPOSE.CELLS IS A PROGRAMMING API that allows developers to create, manipulate and convert Microsoft Excel spreadsheet files from within their own applications. Its powerful features make it easy to convert worksheets and charts to graphics or save reports to PDF.

A flexible API for simple and complex spreadsheet programming.

Aspose.Cells speeds up working with Microsoft Excel files. The API is a flexible tool for simple tasks such as file conversion, as well as complex tasks like building models. Developers control page layout, formatting, charts and formulas. They can read and write spreadsheet files and save out to a wide variety of image and text file formats.

Fast and reliable, Aspose.Cells saves time and effort compared to using Microsoft Office Automation.



G2 =LINEST(E2:E12,A2:D12,TRUE,TRUE)

	A	B	C	D	E	F	G
1	Floor Space (x1)	Offices (x2)	Entrances (x3)	Age (x4)	Assessed Value (y)		
2	2310	2	2	20	142000		-264.334
3	2333	2	2	12	144000		13..26801
4	2366	3	1.5	33	151000		0.996748
5	2379	3	2	43	150000		
6	2402	2	3	53	139000		
7	2425	4	2	23	169000		

Aspose.Cells lets developers work with data sources, formatting, even formulas.

Common Uses

- Building dynamic reports on the fly.
- Creating Excel dashboards with charts and pivot tables.
- Rendering and printing spreadsheets and graphics with high fidelity.
- Exporting data to, or importing from, Excel spreadsheets.
- Generating, manipulating and editing spreadsheets.
- Converting spreadsheets to images or other file formats.

Key Features

- A complete spreadsheet manipulation solution.
- Flexible data visualization and reporting.
- Powerful formula engine.
- Complete formatting control.

Supported File Formats

XLS, XLSX, XLSM, XMPS, XLTX, XLTM, ODS, SpreadsheetML, tab delim., CSV, TXT, PDF, HTML, and many image formats including TIFF, JPEG, PNG and GIF.

Format support varies across platforms.

Platforms

Pricing Info					
	Standard	Enhanced		Standard	Enhanced
Developer Small Business	$999	$1498	Site Small Business	$4995	$7490
Developer OEM	$2997	$4494	Site OEM	$13986	$20972

The pricing info above is for .NET: prices for other platforms may differ. For the latest, contact sales.

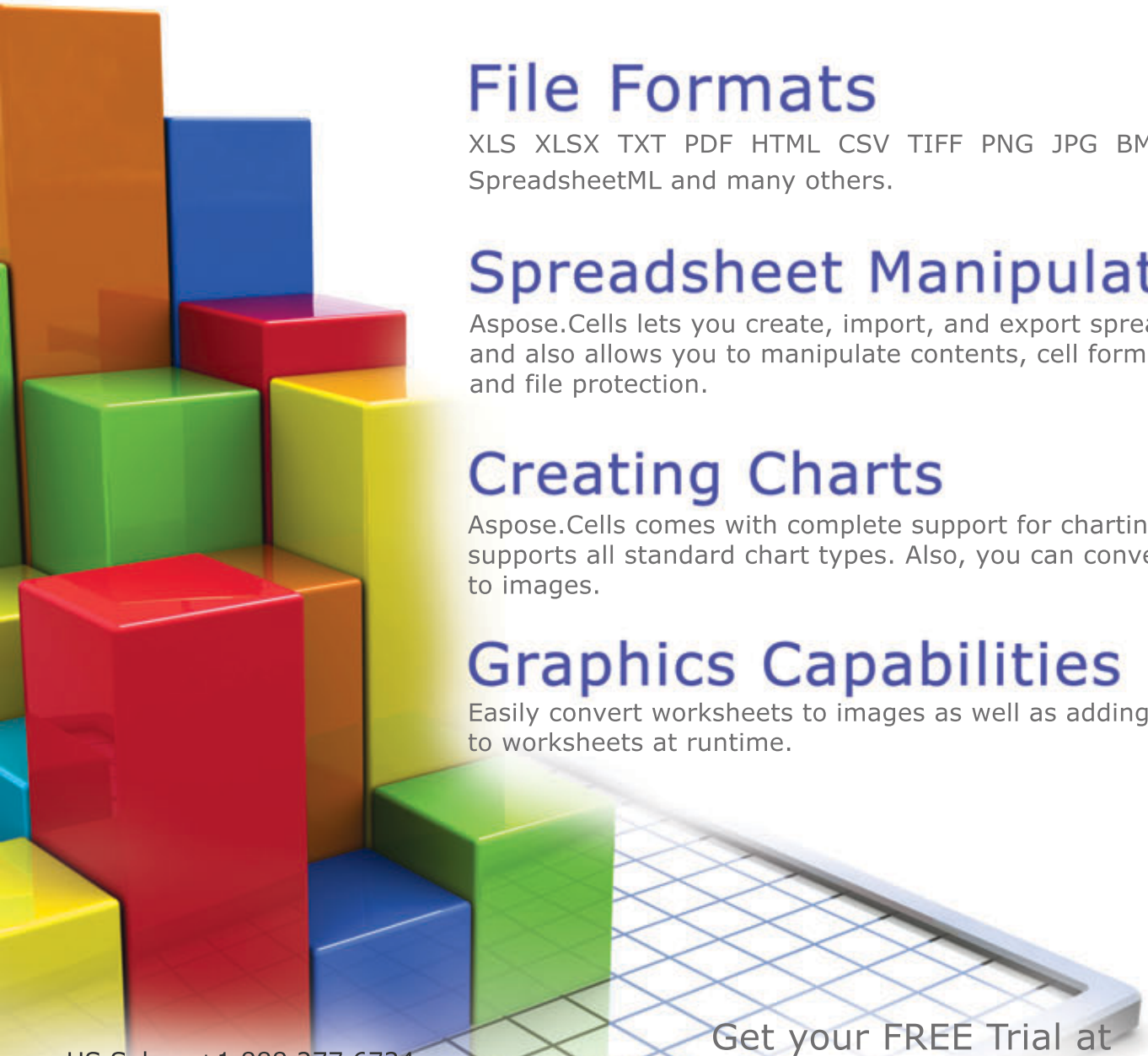
www.aspose.com

US: +1 888 277 6734
sales@aspose.com

EU: +44 141 416 1112
sales.europe@aspose.com

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

Aspose.Cells for
.NET, Java, Cloud & Android



File Formats

XLS XLSX TXT PDF HTML CSV TIFF PNG JPG BMP SpreadsheetML and many others.

Spreadsheet Manipulation

Aspose.Cells lets you create, import, and export spreadsheets and also allows you to manipulate contents, cell formatting, and file protection.

Creating Charts

Aspose.Cells comes with complete support for charting and supports all standard chart types. Also, you can convert charts to images.

Graphics Capabilities

Easily convert worksheets to images as well as adding images to worksheets at runtime.

Get your FREE Trial at
http://www.aspose.com

US Sales: +1 888 277 6734
FAX: +1 866 810 9465
sales@aspose.com

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

ASPOSE
Your File Format APIs

No Office Automation

Aspose.Cells does not require Microsoft Office to be installed on the machine in order to work.

Aspose.Words

Program with word processing documents independently of Microsoft Word

- Solution for document creation, manipulation and conversion.
- Advanced mail merge functionality.

ASPOSE.WORDS IS AN ADVANCED PROGRAMMING API that lets developers perform a wide range of document processing tasks with their own applications. Aspose.Words makes it possible to generate, modify, convert, render and print documents without Microsoft Office Automation. It provides sophisticated and flexible access to, and control over, Microsoft Word files.

Generate, modify, convert, render and print documents without Microsoft Office Automation.

Aspose. Words is powerful, user-friendly and feature rich. It saves developers time and effort compared to using Microsoft Office Automation and makes gives them powerful document management tools.

Aspose.Words makes creating, changing and converting DOC and other word processing file formats fast and easy.

	Table			
	Column 1	**Column 2**	**Column 3**	**Column 4**
Row 1	Cell 1	Cell 2	Cell 3	Cell 4
Row 2	Cell 1	Cell 2	Cell 3	
Row 3	Cell 1	Cell 2		

Aspose.Words has sophisticated controls for formatting and managing tables and other content.

Common Uses

- Generating reports with complex mail merging; mail merging images.
- Populating tables and documents with data from a database.
- Inserting formatted text, paragraphs, tables and images into Microsoft Word documents.
- Adding barcodes to documents.
- Inserting diagrams and watermarks into Word documents.
- Formatting date and numeric fields.

Key Features

- A complete Microsoft Word document manipulation solution.
- Extensive mail merge features.
- Complete formatting control.
- High-fidelity conversion, rendering and printing.

Supported File Formats

DOC, DOCX, ODT, OOXML, XML, HTML, XHTML, MHTML, EPUB, PDF, XPS, RTF, and a number of image formats, including TIFF, JPEG, PNG and GIF.

Format support varies across platforms.

Platforms

Pricing Info					
	Standard	Enhanced		Standard	Enhanced
Developer Small Business	$999	$1498	Site Small Business	$4995	$7490
Developer OEM	$2997	$4494	Site OEM	$13986	$20972

The pricing info above is for .NET: prices for other platforms may differ. For the latest, contact sales.

www.aspose.com

US: +1 888 277 6734
sales@aspose.com

EU: +44 141 416 1112
sales.europe@aspose.com

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

Case Study: Aspose.Words for .NET

Lulu helps authors, publishers, businesses, and educators publish and sell print on demand books and ebooks. Why do they use Aspose?

LULU IS A TECHNOLOGY COMPANY THAT PROVIDES AN OPEN PUBLISHING PLATFORM where customers from all over the world can create, publish and sell print-on-demand books, ebooks, photobooks and calendars.

The basic function of Lulu's publishing platform is to receive manuscripts from customers and send them to printers for printing. The printers receive the manuscripts in PDF format but that is not always its original format.

Customers can submit manuscripts in any number of formats: many use Microsoft Word. Lulu's publishing platform converts incoming manuscripts to PDFs that can then be sent to the printer. The conversion is automatic: the document comes in, is converted and goes off to print withouthuman intervention or review.

Updating the Platform

Lulu has been running for several years. The original conversion platform depended on Microsoft Automation for converting DOC files to PDFs. As the business grew and had to accommodate a much higher number of manuscripts, some problems with the existing platform became apparent.

- It did not scale,
- it did not support the latest Microsoft Word document formats, and
- it was not robust.

Looking for a Solution

> It took only 10 lines of code to integrate Aspose.Words for .NET into the new solution.

The company decided to build a new platform using components that could support their continued growth.

Lulu's engineering team tested Aspose.Words for .NET alongside the existing Microsoft Automation system and other applications. Each solution had its strengths and weaknesses but in the end, Aspose.Words for .NET won because

- it took only 10 lines of code to integrate it into the new platform,
- it is robust and scalable,
- it supports all the file formats that Lulu needs, and
- the licensing structure is straight-forward and cheaper over time than other solutions.

Outcome

The result was a product that can take any Microsoft Word document that a customer submits, regardless of how the customer may have embellish their manuscript, and convert it to a PDF file that can be printed anywhere.

This is an extract from a case study on our website. For the full version, go to: www.aspose.com/corporate/customers/case-studies.aspx

Customers create manuscripts that can be printed by any printer.

GroupDocs™
work faster. together..

http://groupdocs.com
sales@groupdocs.com

POWERFUL DOCUMENT MANAGEMENT LIBRARIES AND CLOUD APIS FOR YOUR WEB AND MOBILE APPLICATIONS

.NET, JAVA, CLOUD APIS

GroupDocs Viewer

True text, high-fidelity embedded document viewer with support for over 50 file formats.

GroupDocs Signature

Electronic signature capture API that gives your apps legally binding e-signature capabilities.

GroupDocs Annotation

A powerful API that lets you annotate Microsoft Office, PDF and other documents within your own applications.

GroupDocs Assembly

Incorporates data entered by users in online forms into PDF and Microsoft Office documents using merge fields.

GroupDocs Conversion

Universal document converter with an independent engine for fast conversion between more than 50 file formats.

GroupDocs Comparison

A diff view API that allows end users to quickly find differences between two revisions of a document.

Sales: +1 (214) 329-9760

Powerful Document Management Solutions for Your Web and Mobile Applications

GroupDocs offers professional stand-alone .NET & Java libraries along with cloud APIs that allow end-users to view, annotate, convert, e-sign, assemble and compare documents and images of more than 35 file formats within your own web and mobile applications. Key benefits include:

- All GroupDocs APIs are 100% independent, and don't require any 3rd party software installation.
- Being extremely lightweight, .NET & Java libraries can be integrated with just a single DLL.
- Cloud APIs are supported by SDKs to help developers on .NET, Java, JavaScript, PHP, Python and Ruby seamlessly integrate GroupDocs solutions into any web or mobile apps.
- No need for client-side installation. End-users can work with documents from any web-enabled device and modern web-browser.
- All GroupDocs' products come with a 30-day fully-functional trial and free support during the integration period.

Stand-Alone .NET & Java Libraries Pricing

GroupDocs .NET & Java licenses are based on the number of developers and the number of locations where the components will be used:

	Developer Small Business	Developer OEM	Site Small Business	Site OEM
License for one developer	✓	✓		
Licenses for up to 10 developers			✓	✓
Use derived work at one location	✓			
Use derived work at up to 10 locations			✓	
Royalty free/deploy to unlimited locations		✓		✓
Discount applied to multiple purchases	✓	✓	✓	✓
Can be used to create unlimited applications	✓	✓	✓	✓
Updates and hotfixes for one year	✓	✓	✓	✓
Free technical support	✓	✓	✓	✓
Price	**$2,499**	**$7,497**	**$9,996**	**$29,988**

Cloud API Pricing

GroupDocs' cloud APIs use a different licensing model. Instead of licenses, they are charged by use: the number of calls made to the API. To find out more about our cloud API pricing, please visit our website: www.groupdocs.com

GroupDocs Viewer

A powerful document viewer API that allows you to display over 35 document formats in your web or mobile application. The viewer can both rasterize documents and convert them to SVG+HTML+CSS, delivering true-text high-fidelity rendering.
Supported file formats include: Microsoft Office, Visio, Project and Outlook documents, PDFs, AutoCAD, image files (TIFF, JPG, BMP, GIF, TIFF, etc.) and more.

GroupDocs Signature

GroupDocs Signature API is an easy way to give your apps legally binding e-signature capabilities. Your users are then able to get documents signed electronically using only a web-browser.
The API gives developers access to sophisticated online signature features, from e-signature capture control and different signing workflows, to reminder management, contact management and signer roles.

GroupDocs Annotation

With support for over 35 file formats, this API allows your app users to annotate documents of all common business formats, including Microsoft Office and PDF. And thanks to the advanced document management options, users can store, share, print, download and export the annotated documents easily - all from within your own application.

GroupDocs Assembly

GroupDocs Assembly automatically incorporates data submitted through online forms into existing document templates in PDF or Microsoft Word formats. For each completed form a new custom document is generated. The API lets you build document assembly solutions without getting into the details of working with templates, fields and merging data.

GroupDocs Conversion

GroupDocs Conversion API allows end users to convert back and forth between over 25 document formats within your own application. It supports all Microsoft Office document formats as well as PDF, HTML and common image file formats (TIFF, JPEG, GIF, PNG, BMP). Your users can convert documents one by one on the fly, or add several documents at a time to a conversion queue.

GroupDocs Comparison

A document comparison API that allows users to quickly and easily find differences between two revisions of a document right in your web or mobile app. It merges two uploaded documents into a single one and displays it, highlighting differences with the redline view approach - similar to the Microsoft Word change tracking feature, but online. Works with Microsoft Word, Excel, and PowerPoint documents, as well as Adobe Acrobat PDF files.

Adding File Conversion and Manipulation to Business Systems

How often do people in your organization complain that they can't get information in the file format and layout they want? Converting documents from one format to another without losing layout and formatting should be simple, but it can be frustrating for both users and developers.

EXTRACTING DATA FROM A DATABASE AND DELIVERING IT TO THE SALES TEAM AS A REPORT, complete with charts and corporate branding, is fine. Until the sales team says that they want it as a Microsoft Excel file, and could you add a dashboard?

Using information from online forms in letters that can are printed and posted is easy. But what if you also want to add tracking barcodes and archive a digital copy as a PDF?

Ensuring that your business system supports all the different Microsoft Office file formats your users want can be difficult. Sometimes the native file format support of your system lets you down. When that is the case, use tools that extend that capability. A good tool can save you time and effort.

Document Conversion Options

Building your own solution: Time-consuming and costly, this option is only sensible if the solution you develop is central to your business.

Using Microsoft Office Automation: Microsoft Office Automation lets you use Microsoft Office programs server-side. It is not how the Office products were designed to be used. It can work well but you might notice issues with the stability, security and speed of the system, as well as cost.

> Aspose creates APIs that work independently of Microsoft Office Automation.

Using an API: The API market has lots of free and commercial solutions, some very focused, some feature-rich. An API integrates with your code and gives you access to a range of new features.

Look to Aspose

Aspose are API experts. We create APIs, components and extensions that work independently of Microsoft Automation to extend a platform's native file format manipulation capabilities.

Aspose have developed APIs for .NET, Java, Cloud and Android that lets developers convert, create and manipulate Microsoft Office files – Microsoft Word, Excel, PowerPoint, Visio and Project – and other popular business formats, from PDFs and images to emails. We also have APIs for working with images, barcodes and OCR. The APIs are optimised for stability, speed and ease of use. Our APIs save users weeks, sometimes months, of effort.

Finding the Right Tool

To find the product that's right for you, take a systematic approach:

- List must-have and nice-to-have features.
- Research the market.
- Ask for recommendations.
- Select a few candidates .
- Run trials.
- Evaluate
  - ease of use,
  - support and documentation,
  - performance, and
  - current and future needs.

Aspose.BarCode

A complete toolkit for barcode generation and recognition

- Generate barcodes with customer defined size and color.
- Recognize a large number of barcode types from images.

ASPOSE.BARCODE IS A ROBUST AND RELIABLE BARCODE GENERATION AND RECOGNITION API that allows developers to add barcode generation and recognition functionality to their applications quickly and easily.

Aspose.BarCode supports most established barcode specifications. It can export generated barcodes to multiple image formats, including BMP, GIF, JPED, PNG and TIFF.

Aspose.BarCode gives you full control over every aspect of the barcode image, from background and bar color, through image quality, rotation angle, X-dimension, captions, and resolution.

Robust and reliable barcode generation and recognition.

Aspose.BarCode can read and recognize most common 1D and 2D barcodes from any image and at any angle. Filters help developers clean up difficult to read images to improve recognition.

Aspose.BarCode offers a large number of symbologies and formatting options.

Common Uses

- Generating and recognizing barcode images.
- Printing barcode labels.
- Enhancing workflow by adding barcode functionality.
- Using recognition functions to drive real-life work processes.

Key Features

- Barcode generation and recognition.
- Comprehensive support for 1D and 2D symbologies.
- Image processing for improved recognition.

Supported File Formats

JPG, TIFF, PNG, BMP, GIF, EMF, WMF, EXIP and ICON.

Format support varies across platforms.

Supported Barcodes

Linear: EAN13, EAN8, UPCA, UPCE, Interleaved2of5, Standard2of5, MSI, Code11, Codabar, EAN14(SCC14), SSCC18, ITF14, Matrix 2 of 5, PZN, Code128, Code39 Extended, Code39 Standard, OPC, Code93 Extended, Code93 Standard, IATA 2 of 5, GS1Code128, ISBN, ISMN, ISSN, ITF6, Pharmacode, DatabarOmniDirectional, VIN, DatabarTruncated, DatabarLimited, DatabarExpanded, PatchCode, Supplement **2D**: PDF417, MacroPDF417, DataMatrix, Aztec, QR, Italian Post 25, Code16K, GS1DataMatrix **Postal**: Postnet, Planet, USPS OneCode, Australia Post, Deutsche Post Identcode, AustralianPosteParcel, Deutsche Post Leticode, RM4SCC, SingaporePost, SwissPostParcel

Platforms

Pricing Info					
	Standard	Enhanced		Standard	Enhanced
Developer Small Business	$599	$1098	Site Small Business	$2995	$5490
Developer OEM	$1797	$3294	Site OEM	$8386	$15372

The pricing info above is for .NET: prices for other platforms may differ. For the latest, contact sales.

www.aspose.com

US: +1 888 277 6734
sales@aspose.com

EU: +44 141 416 1112
sales.europe@aspose.com

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

Aspose.Email

Work with emails and calendars without Microsoft Outlook

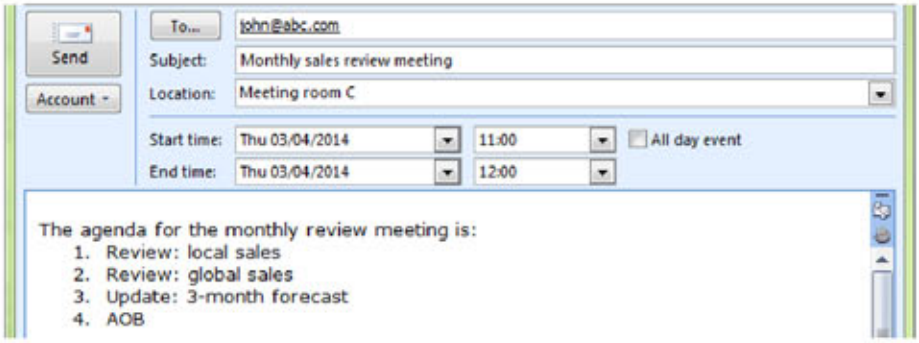
- Complete email processing solution.
- Message file format support.

ASPOSE.EMAIL IS AN EMAIL PROGRAMMING API that allows developers to access and work with PST, EML, MSG and MHT files. It also offers an advanced API for interacting with enterprise mail systems like Exchange and Gmail.

Aspose.Email can work with HTML and plain text emails, attachments and embedded OLE objects. It allows developers to work against SMTP, POP, FTP and Microsoft Exchange servers. It supports mail merge and iCalendar features, customized header and body, searching archives and has many other useful features.

Aspose.Email works with HTML and plain text emails, attachments and embedded OLE objects.

Aspose.Email allows developers to focus on managing email without getting into the core of email and network programming. It gives you the controls you need.



Aspose.Email lets your applications work with emails, attachments, notes and calendars .

Common Uses

- Sending email with HTML formatting and attachments.
- Mail merging and sending mass mail.
- Connecting to POP3 and IMAP mail servers to list and download messages.
- Connecting to Microsoft Exchange Servers to list, download and send messages.
- Create and update tasks using iCalendar.
- Load from and save messages to file or stream (EML, MSG or MHT formats).

Key Features

- A complete email processing solution.
- Support for MSG and PST formats.
- Microsoft Exchange Server support.
- Complete recurrence pattern solution.

Supported File Formats

MSG, MHT, OST, PST, EMLX, TNEF, and EML.

Format support varies across platforms.

Platforms

Pricing Info					
	Standard	Enhanced		Standard	Enhanced
Developer Small Business	$599	$1059	Site Small Business	$2995	$5490
Developer OEM	$1797	$3294	Site OEM	$8386	$15372

The pricing info above is for .NET: prices for other platforms may differ. For the latest, contact sales.

www.aspose.com

US: +1 888 277 6734
sales@aspose.com

EU: +44 141 416 1112
sales.europe@aspose.com

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

Aspose.Pdf

Create PDF documents without using Adobe Acrobat

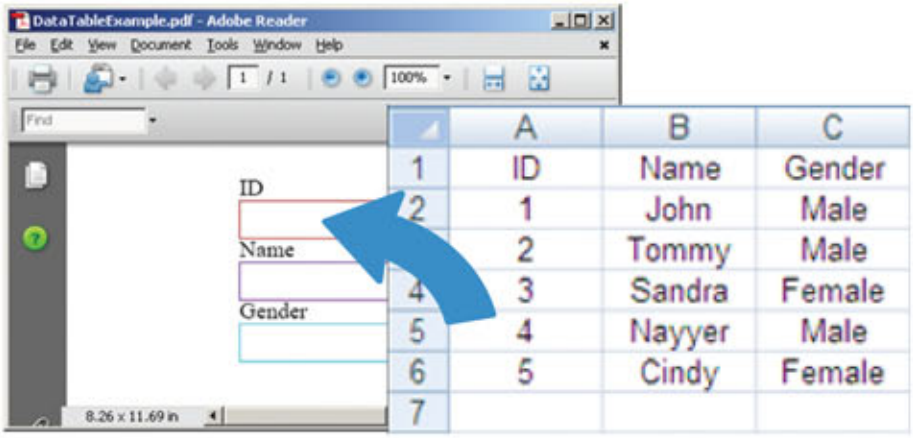
- A complete solution for programming with PDF files.
- Work with PDF forms and form fields.

ASPOSE.PDF IS A PDF DOCUMENT CREATION AND MANIPULATION API that developers use to read, write and manipulate PDF documents without using Adobe Acrobat. Aspose.Pdf is a sophisticated product that integrates with your application to add PDF capabilities.

Aspose.Pdf offers a wealth of features that lets developers compress files, create tables, work with links, add and remove security, handle custom fonts, integrate with external data sources, manage bookmarks, create table of contents, create forms and manage form fields.

Read, write and manipulate PDF documents independently of Adobe Acrobat.

It helps developers add, work with attachments, annotations and PDF form data, add, replace or remove text and images, split, concatenate, extract or inset pages, and print PDF documents.



Aspose.Pdf can be used to automatically complete PDF forms with external data.

Common Uses

- Creating and editing PDF files.
- Inserting, extracting, appending, concatenating and splitting PDFs.
- Working with text, images, tables, images, headers, and footers.
- Applying security, passwords and signatures.
- Working with forms and form fields.

Key Features

- PDF creation from XML or XLS-FO documents.
- PDF form and field support.
- Advanced security and encryption.
- High-fidelity printing and conversion.
- Supported File Formats
- PDF, PDF/A, PDF/A_1b, PCL, XLS-FO, LaTeX, HTML, XPS, TXT and a range of image formats.

Format support varies across platforms.

Platforms

Pricing Info					
	Standard	Enhanced		Standard	Enhanced
Developer Small Business	$799	$1298	Site Small Business	$3995	$6490
Developer OEM	$2397	$3894	Site OEM	$11186	$18172

The pricing info above is for .NET: prices for other platforms may differ. For the latest, contact sales.

www.aspose.com

US: +1 888 277 6734
sales@aspose.com

EU: +44 141 416 1112
sales.europe@aspose.com

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

Banckle
Customer Service 2.0

http://banckle.com/
sales@banckle.com

Browser Based

Powerful APIs

Device Compatibility

Customer Service Tools that Run in a Web-Browser

Banckle's online customer service tools help companies large and small engage with customers. Just log in and go: no plugin or software installation necessary. Platform, OS and browser independent.

Build powerful SaaS apps with Banckle's robust Cloud APIs.

Contact **sales@banckle.com** for API pricing information and an extended 30 day free trial. Alternatively, visit **banckle.com** and use promo code **MiniMag2014** at checkout for a 20% discount.

Banckle.com - Customer Service 2.0

Customer Service Apps that Help Grow Your Business

Banckle's professional web apps help companies engage with customers.

Banckle Chat

Chat live with your customers to give them the help they want, when they need it. Embeddable HTML makes it easy to integrate Banckle's live chat service into your website.

Prices start from $6.30/month.

Banckle Meeting

Give your customers a great online meeting experience with an intuitive interface. Share screen, presentations, work with a whiteboard and use video conferencing from a web browser.

Prices start from $10.50/month.

Banckle CRM

Keep the team up to date with projects and customer contacts. Keep an eye on the pipeline, manage tasks and log contacts in one central place to improve team work and customer focus.

Prices start from $4.20/month.

Banckle Helpdesk

Stay on top of your customers' issues with an online service desk and ticketing system. Never lose a ticket but let the team collaborate to deliver the best possible customer support.

Prices start from $11.00/month.

Banckle Campaign

Find out how effective your email campaigns are and keep in touch with your customers. Design, test and send campaigns, manage mailing lists and learn from campaign reports.

Prices start from $9.80/month.

Banckle Email

Manage email from an affordable, secure and intuitive online platform. Use all the email features you are used to: attachments, folders and simple account administration.

Prices start from $13.30/month.

Banckle Total

All Banckle's customer service tools in one package, Banckle Total helps you take customer support to the next level.

Subscriptions start at $31.00/month.

Try Our APIs

Want to build your own solution?, Consider our RESTful API's

Contact **sales@banckle.com** for API pricing information and an extended 30 day free trial. Alternatively, visit **banckle.com** and use promo code **MiniMag2014** at checkout for a 20% discount.

Aspose.Slides

Work with presentations without using Microsoft PowerPoint

- Complete solution for working with presentation files.
- Export presentations and slides to portable or image formats.

ASPOSE.SLIDES IS A FLEXIBLE PRESENTATION MANAGEMENT API that helps developers read, write and manipulate Microsoft PowerPoint documents. Slides and presentations can be saved to PDF, HTML and image file formats without Microsoft Office Automation.

Aspose.Slides offers a number of advanced features that make it easy to perform tasks such as rendering slides, exporting presentations, exporting slides to SVG and printing. Developers use Aspose.Slides to build customizable slide decks, add or remove standard graphics and automatically publish presentations to other formats.

> Aspose.Slides gives you the tools you need to work with presentation files.

Aspose.Slides gives developers the tools they need to work with presentation files. It integrates quickly and saves time and money.

Aspose.Slides has advanced features for working with every aspect of a presentation.

Common Uses

- Creating new slides and cloning existing slides from templates.
- Handling text and shape formatting.
- Applying and removing protection.
- Exporting presentations to images and PDF.
- Embedding Excel charts as OLE objects.
- Generate presentations from database.

Key Features

- A complete presentation development solution.
- Control over text, formatting and slide elements.
- OLE integration for embedding external content.
- Wide support for input and output file formats.

Supported File Formats

PPT, POT, PPS, PPTX, POTX, PPSX, ODP, PresentationML, XPS, PDF and image formats including TIFF and JPG.

Format support varies across platforms.

Platforms

Pricing Info					
	Standard	Enhanced		Standard	Enhanced
Developer Small Business	$799	$1298	Site Small Business	$3995	$6490
Developer OEM	$2397	$3894	Site OEM	$11186	$18172

The pricing info above is for .NET: prices for other platforms may differ. For the latest, contact sales.

www.aspose.com

US: +1 888 277 6734
sales@aspose.com

EU: +44 141 416 1112
sales.europe@aspose.com

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

Support Services

Get the assistance you need, when you need it, from the people who know our products best.

- Use experienced Aspose developers for your projects
- Get the level of support that suits you and your team

NO ONE KNOWS OUR PRODUCTS AS WELL AS WE DO. We develop them, support them and use them. Our experience is available to you, whether you want us to develop a solution for you, or you just need a little help to solve a particular problem.

Consulting

Aspose's developers are expert users of Aspose APIs. They understand how to use our products and have hands-on experience of using them for software development. Aspose's developers are skilled not just with Aspose tools but in a wide range of programming languages, tools and techniques.

Aspose's file format experts are here to help you with a project or your support questions

When you need help to get a project off the ground, Aspose's developers can help.

Work with the most experienced Aspose developers in the world.

Consulting Benefits

- Use Aspose engineers to work on your products
- Get peace of mind from a fully managed development process
- Get a custom-built solution that meets your exact needs

Support Options

Free

Everyone who uses Aspose products have access to our free support. Our software developers are on stand-by to help you succeed with your project, from the evaluation to roll-out of your solution.

Priority

If you want to know when you'll hear back from us on an issue, and know that your issue is prioritized, Priority Support is for you. It provides a more formal support structure and has its own forum that is monitored by our software engineers.

Enterprise

Enterprise customers often have very specific needs. Our Enterprise Support option gives them access to the product development team and influence over the roadmap. Enterprise Support customers have their own, dedicated issue tracking system.

Pricing Info
Each consulting project is evaluated individually; no two projects have exactly the same requirements.
To see the Priority and Enterprise support rates, refer to the product price list, or contact our sales team.

We're Here to Help YOU

Aspose has 4 Support Services to best suit your needs

Free Support	Support Forums with no Charge
Priority Support	24 hour response time in the week, issue escalation, dedicated forum
Enterprise Support	Communicate with product managers, influence the roadmap
Sponsored Support	Get the feature you need built now

Technical Support is an issue that Aspose takes very seriously. Software must work quickly and dependably. When problems arise, developers need answers in a hurry. We ensure that our clients receive useful answers and solutions quickly.

Email • Live Chat • Forums

CONTACT US

US Sales: +1 888 277 6734
sales@aspose.com

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

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