THE MICROSOFT JOURNAL FOR DEVELOPERS

MAY 2014 VOL 29 NO 5

msdn magazine

C#

THE MICROSOFT JOURNAL FOR DEVELOPERS

MAY 2014 VOL 29 NO 5

msdn magazine

C#

An Introduction to C# vNext.....16

COLUMNS

Microsoft

msdn magazine

MAY 2014 VOLUME 29 NUMBER 5

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

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

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

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

List Rental: This publication's subscriber list, as well as other lists from 1105 Media, Inc., is available for rental. For more information, please contact our list manager, 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

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

Visit Us at TechEd Booth #1934

Melissa Data.
Architecting data quality success.

MELISSA DATA®

www.MelissaData.com 1-800-MELISSA

Editor's Note

MICHAEL DESMOND

Building Bridges

The Microsoft Build 2014 conference took place at the Moscone Center in San Francisco early last month and set the tone for the upcoming year in application development across the Microsoft ecosystem. And what developers saw was a Microsoft eager to meet them where they live—be it on the desktop, in the cloud, on tablets or phones, or even on the Xbox gaming console and competing device platforms such as iOS and Android. Microsoft is building bridges in 2014, and the impact of that effort will color development for years to come.

That message was sent loud and clear in the first hour of the opening day keynote, when David Treadwell, corporate vice president of the Operating Systems Group at Microsoft, unveiled Microsoft's Universal Windows apps strategy. The approach enables developers to maintain a single code base that can target Windows 8 desktops and tablets, Windows Phone handhelds, and even the Xbox One entertainment platform. Universal Windows apps don't promise full "write once, run everywhere" capability—devs will usually tweak the UI for each target—but it does create huge potential value for developers coding to the Windows Runtime.

Microsoft's bridge-building efforts span gaps beyond the Windows family as well. New Microsoft CEO Satya Nadella spotlighted the efforts of partners like Xamarin, PhoneGap and Unity to enable efficient, cross-platform development from Visual Studio and the Microsoft .NET Framework. Xamarin co-founder Miguel de Icaza, for example, took the stage during the day-one keynote to demonstrate how the Xamarin add-on for Visual Studio extends C# development to iOS and Android.

Perhaps most notable at Build were the bridges Microsoft worked to build for incumbent developers. Three years after unveiling its Windows Runtime strategy at the Build conference in 2011, Microsoft made a point this year to strongly affirm its commitment to the .NET Framework. As Microsoft Technical Fellow Anders Hejlsberg told a gathering of journalists during the conference, "We are going all-in on .NET."

The commitment to the .NET Framework was evident in the launch of the open source .NET Compiler Platform (Project "Roslyn") and the forthcoming update of the managed C# programming language that is the subject of this month's lead feature by Mark Michaelis (p. 16). It's also evident in the release of .NET Native, which compiles C# to native machine code to yield quicker start up and reduced memory footprint compared to apps based on managed C#. Microsoft also launched the .NET Foundation, an umbrella organization to help shepherd the growing fleet of open source technologies in the .NET Framework.

Perhaps most notable at Build were the bridges Microsoft worked to build for incumbent developers.

As is the case with almost every Build conference, this year's get-together created nearly as many questions as it answered. But with Microsoft's renewed commitment to the .NET Framework, and impressive support for cross-platform development both within and beyond the Microsoft ecosystem—not to mention what seems to be a fully committed embrace of open source—there's a lot to be excited about.

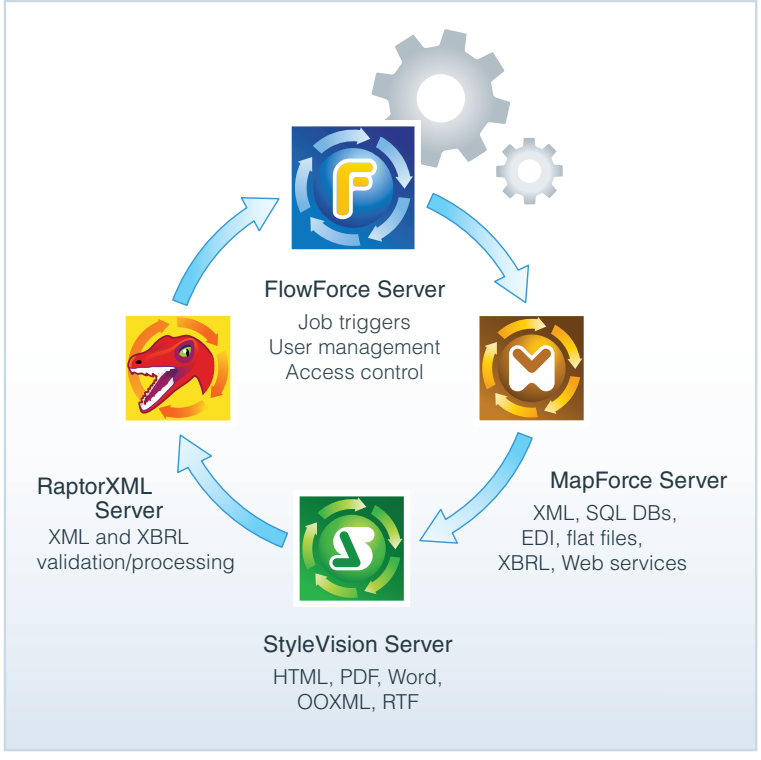
Still looking for answers? You might check out the Visual Studio Live! (bit.ly/1k0vBvJ) event in Chicago, May 5-8. This four-day gathering of developers, software architects and designers offers cutting-edge education on Microsoft development platforms. Master the emerging tools and techniques that are shaping development in the Microsoft ecosystem, and maybe build a few bridges of your own.

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

Manage Information Workflows with Altova® FlowForce® Server

ALTOVA®
FlowForce®
SERVER 2014

Introducing FlowForce Server, the powerful new platform for automating today's multi-step, enterprise-level data mapping, transformation, integration, and reporting tasks. This flexible workflow orchestration tool works with any combination of XML, database, flat file, EDI, Excel, XBRL, and/or Web service data common to many essential business processes.



FlowForce Server is at the center of Altova's new line of cross-platform server products optimized for today's high-performance, parallel computing environments:

- **FlowForce® Server** for orchestrating events, triggers, and the automation of processes
- **MapForce® Server** for automating any-to-any data mapping and aggregation processes
- **StyleVision® Server** for automating business report generation in HTML, PDF, and Word
- **RaptorXML® Server** for hyper-fast validation/processing of XML, XBRL, XSLT, and XQuery

Learn more and download a free trial at www.altova.com/server

BRUNO TERKALY AND
RICARDO VILLALOBOS

Microsoft Azure and Open Source Power Grid Computing

Imagine building your own grid computing platform that leverages Microsoft Azure and a large number of connected devices. The goal is to leverage the excess computing power found in modern browsers, sending each client a small amount of JavaScript code and data to perform a computing job. Upon completion, each device connected to this grid sends the results back to a central server residing in Azure.

There's something like this actually in place—the Search for Extra Terrestrial Intelligence (SETI) project. The search for extraterrestrial life uses a large-scale grid or distributed computing over the Internet. It monitors space for signs of transmissions from alien civilizations by analyzing electromagnetic radiation in the microwave spectrum. It's a good example of the power of grid computing.

General-Purpose Grid

In this month's column, we'll create a more general-purpose grid computing system. This will let us send specific code and data we want executed on each grid node. For this project, each client browser will receive a chunk of JavaScript along with the information to be processed. This lets us more precisely control the task executed in the browser. The example we'll present solves many general-purpose computing problems that might come up in the context of grid computing.

The genesis of this work came out of the participation of Microsoft in one of the world's largest hackathons, Tech Crunch Disrupt 2013. Microsoft took third place out of 280 teams. You can see the entire solution at tcrn.ch/Oklchx.

The challenge at a competition like this is you only have two days to complete a project before the judges come in and shoot you down. Besides dealing with sleep deprivation, you have to leverage as many prebuilt components as possible to complete the project on time. Most, if not all, of the technology used in the competition was based on open source software running in Azure. The open source technologies used included Jade, Express, Socket.io, Bootstrap, jQuery and Node.js.

Web Sockets

We relied heavily on the now ubiquitous Web Sockets standard. Web Sockets are part of the HTML5 initiative. They provide a full duplex bidirectional connection over which you can transmit messages between client and server. Web Sockets enable a standardized approach for the server to send content to the browser without being explicitly asked by the client.

This let us exchange messages back and forth while keeping the connection open—creating full communication and orchestration, which is a necessary capability for a grid computing system. Today's modern browsers such as Firefox 6, Safari 6, Google Chrome 14, Opera 12.10 and Internet Explorer 10 (and later) universally support Web Sockets.

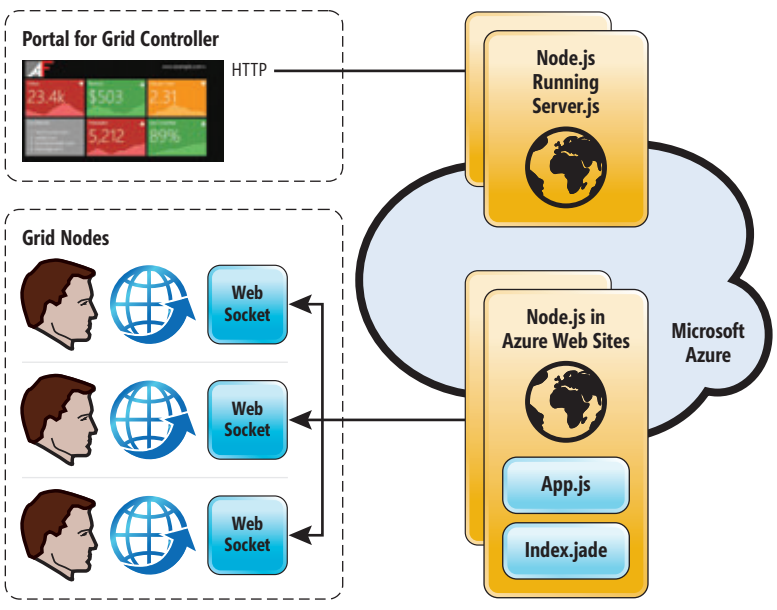


Figure 1 **High-Level Grid Architecture**

Role of Web Sockets

Web Sockets start working when the client sends a Web Socket handshake request to the server in the form of an HTTP GET request. With Web Sockets, what follows the handshake doesn't conform to the standard HTTP protocol. Data text frames in full duplex are sent back and forth, with each text frame representing a payload accompanied by a small header. You can split larger messages across multiple data frames.

The Web Socket plumbing tries to detect if there's a user agent configured, which would let you establish a persistent communication tunnel. In our implementation, the user agent is simply a field in the HTTP header used to send a special HTTP request that basically says, "Switch to Web Sockets." In this article, we'll use Web Sockets to send executable JavaScript and data to each Web client. Once the job is complete, we'll use the Web

Sockets to send computational results back to the Node.js server. This is the key part of our architecture we'll explain later.

Running the entire project yourself is quite easy. You can view a brief video that shows the project in action at 1drv.ms/1d79pjo. Before watching the video, you can grab all the code from GitHub at bit.ly/1mgWWwc. Setting up the project to run is straightforward with Node.js:

1. Start by installing Node.js from nodejs.org
2. Install Git (git-scm.com) or GitHub (github.com)
3. Clone your fork with a Git clone (bit.ly/1cZ1nZh)
4. Install the Node.js package manager (NPM) in the cloned directory
5. Start NPM to run

You'll need to install the various Node.js packages highlighted in this column. You can download the packages using the NPM at npmjs.org. You can also learn how to install them with a right-click in Visual Studio at bit.ly/OBbtEF. To learn more about using Visual Studio with Node.js, check out Bruno's blog post, "Getting Started with Node.js and Visual Studio" (bit.ly/1gzKkbj).

Focus on App.js

The final solution we created actually has two server-side processes. The first and most obvious server-side process is the one that's breaking the large computing job into smaller pieces and distributing the work and data to connected client browsers. You'll find that code in App.js.

There's a second server-side process that provides a portal experience to managing and viewing the large computing jobs executing on the grid. You'll find that code in Server.js. It provides a real-time dashboard experience, complete with live updating graphs and numbers through a browser (see **Figure 1**). Our column will focus on the App.js code.

Orchestration Details

Node.js provides some surprisingly powerful abstractions that help you assemble an elegant implementation. First, you need to solve the problem of sending a piece of JavaScript code you wish to execute as part of the large grid job. You also need to send some data the Javascript code will use.

You can use the Node.js package Express and Socket.io to accomplish this. It's not enough to just send just one piece of JavaScript code and data to the browser. You still need a way to execute the code against the data and send the result back to the server. You can resolve this with the Index.jade module. This means there's a second piece of JavaScript code to manage executing the grid code itself.

Three node packages (along with some supporting packages) greatly simplify implementing this architecture. For example, the express package is a popular package that helps with URL routes, handling requests and views. It also simplifies things such as parsing payloads, cookies and storing sessions.

Another powerful package is Socket.io, which abstracts away Web Sockets and includes convenient features such as broadcasts and multicasts. Socket.io lets you set up bidirectional communication using syntactically identical JavaScript code on both the

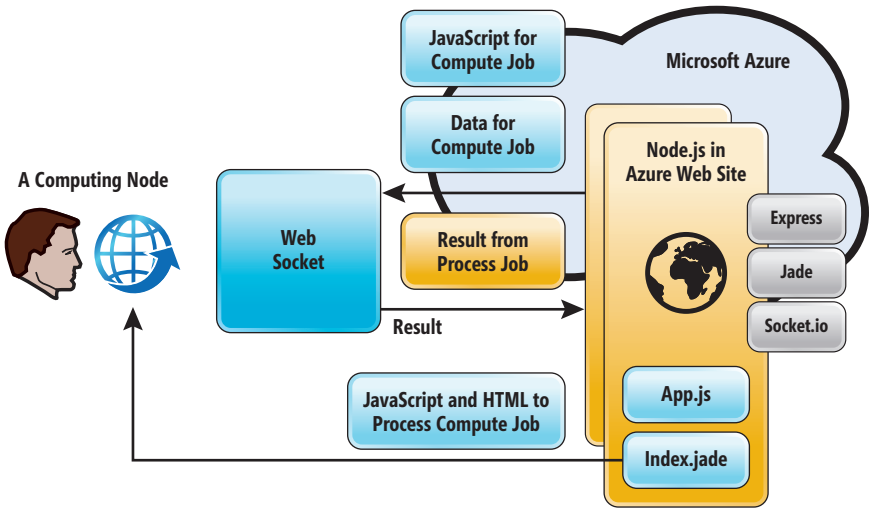


Figure 2 **The Bidirectional Communication of Grid Architecture**

server and the browser. Socket.io manages the JavaScript that runs in the browser and the server. This is precisely what we believe makes Node.js great. There's no mental context switching with writing JavaScript that runs on the server versus the client.

Node.js is tightly integrated with Jade, which streamlines the process of creating a Web interface. Jade provides a template-based approach to creating the HTML, in addition to containing the orchestration JavaScript code that manages the communication between server and browser (client).

Taken together, all of the packages referenced in **Figure 2** will dramatically reduce the amount of code you have to write. A good Node.js developer understands the language and the built-in capabilities. A great Node.js developer is familiar with the various packages and is skilled at using them efficiently. Do yourself a favor and familiarize yourself with the Node.js Packaged Modules library at npmjs.org.

Bidirectional Logic

Ultimately, the orchestration between client and server is nothing more than the state machine bidirectional logic. For example, the client might be in a waiting state for the JavaScript code or it might be in a waiting state for the data to be received.

The server side will have corresponding states, such as the sending JavaScript state or the sending data state. You'll notice statements in the Node.js code, such as "Socket.on("some state")," indicating the server is waiting to receive a magic string to trigger a state change (see **Figure 3**). Then it can respond appropriately to that event.

Let's begin by examining the setup code for the Node.js server-side process. The workflow begins when the server opens a port and waits for connections. Both Express and Socket.io let the server listen for incoming connections of browsers on port 3,000:

```
// Create a Web server, allowing the Express package to
// handle the requests.
var server = http.createServer(app);

// Socket.io injects itself into HTTP server, handling Socket.io
// requests, not handled by Express itself.
var io = socketio.listen(server);
```

Send JavaScript to the Browser

Once the connection is established, the server waits for a message from the client that indicates the client is ready to receive some JavaScript code through the Web Socket connection. The code in line nine in **Figure 4** represents the server waiting for the connection to take place and for the client to send the string "ready for job," indicating to the server the JavaScript should be sent back to the client.

At this point, we're halfway there. The client still needs to request the data to process. The code is some basic trigonometric code to calculate distance between two points using GPS coordinates. You could substitute any JavaScript code you want here.

The code in part two represents the state in which the server waits for the string "ready for data" at line 25 in **Figure 4**. This signals the browser's request for data. The JavaScript previously sent in part one will process this data. The code in part three represents the state in which the client browser has finished computations on the sent data. When the server receives the string results at line 41, it's ready to incorporate that browser's final result for the computing job. At this time, the browser could be sent another job to do more processing, repeating the cycle.

The Jade Engine

Jade is a productive HTML view and templating engine integrated into Node.js. Jade greatly simplifies the markup and JavaScript you write for the browser. **Figure 5** shows the Jade markup language that defines the UI.

First, it simply shows the job progress to the browser. Second, it takes the JavaScript and data sent by the server. This represents the computational job it needs to perform. It executes the job, returning the results back to the server.

If you've ever wondered how to send JavaScript to a browser for execution, **Figure 5** represents the code you'll need to do this. If you want to know more about how Jade works, we recommend this brilliantly simple explanation at jade-lang.com. The bottom line is you can code up a visual interface without all the tricky HTML tags, angle brackets and so on.

Figure 3 **Partial Listing in App.js for Setting up Node.js Packages**

```
// Setup libraries.
var express = require('express');
var routes = require('./routes');
var user = require('./routes/user');
var http = require('http');
var path = require('path');
var socketio = require('socket.io');
var app = express();
var azure = require('azure');
var fs = require('fs')

// Code omitted for brevity.

// Let Jade handle the client-side JavaScript and HTML
app.set('views', __dirname + '/views');
app.set('view engine', 'jade');
```

Figure 4 **Partial Listing of Server-Side Code That Distributes JavaScript and Data to Browsers on the Grid**

```
(001) // Code Part 1
(003) // Wait for the browser to say it's ready for the job.
(005) // If it is, send the JavaScript to the grid node for execution.
(007) // Do the same thing for the data being sent to the browser.
(009) io.on('connection', function(socket) {
(011)   socket.on('ready for job', function() {
(013)     clients++;
(015)     socket.emit('job', 'function process(message){function
isInRange(origin,target,range){function toRad(deg){return deg*Math.PI/180}
function getDistance(origin,target){var R=6371;var delta={lat:toRad(target.
lat-origin.lat),lon:toRad(target.lon-origin.lon)};var start=toRad(origin.
lat);var end=toRad(target.lat);var a=Math.sin(delta.lat/2)*Math.sin(delta.
lat/2)+Math.sin(delta.lon/2)*Math.sin(delta.lon/2)*Math.cos(start)*Math.
cos(end);var c=2*Math.atan2(Math.sqrt(a),Math.sqrt(1-a));return R*c}return
getDistance(origin,target)<range}function parseData(data){var parts=data.
split(",");return{lat:parts[parts.length-1],lon:parts[parts.length-2]}}
var target=parseData(message.body);var origin={lat:37.769578,lon:-
122.403663};var range=5;return isInRange(origin,target,range)?1:0}');
(017)   });

(018)
(021) // Code Part 2  Sending data to the browser for processing
(023) // when 'ready for data event' fires off.
(025)   socket.on('ready for data', function() {
(027)     socket.isClient = true;
(029)     sendDataToSocket(socket);
(031)   });

(032)
(035) // Code Part 3 - retrieving the results of the computation.
(037) // A more thorough implementation will aggregate all the
      // results from all the browsers to solve the large computational
(039) // problem that has been broken into small chunks for each browser.
(041)   socket.on('results', function(message, results) {
(043)     messageCount++;
(045)     crimesInRange += results;
(047)   });

(048)
(051) // Code Part 4 - A basic method to send data to a connected
      // client with a timeout of 77 ms.
(053) function sendDataToSocket(socket) {
(055)   var data = lines.shift();
(057)   lines.push(data);
(059)   setTimeout(function() {
(061)     // To one client, singular
(063)     socket.emit('process', {
(065)                             body: data
(067)     });
(069)   }, 77);
(071) }
```

Figure 5 **Jade Defines the UI**

```
// Part 1
// This UI markup gets translated into real HTML
// before running on the client.

block content
  h1= title
  p This is an example client Web site. Imagine a beautiful Web site
without any advertisements!
  p This page is processing
    span#items
    |  jobs per second.

// Part 2
// This is the client-side JavaScript code.
  script.
    var socket = io.connect();
    var job = function(id, data) {  };

    var createFunction = function(string) {
      return (new Function( 'return (' + string + ')' )());
    }

    var items = 0;
    function calculateWork() {
      $('#items').text(items);
      items = 0;
    }

    setInterval(calculateWork, 1000);

    socket.on('connect', function() {
      socket.emit('ready for job');
    });

    socket.on('job', function(fn) {
      job = createFunction(fn);
      console.log(fn);
      console.log(job);
      socket.emit('ready for data');
    });

    socket.on('process', function(message) {
      var results = job(message);
      items++;
      socket.emit('results', message, results);
      socket.emit('ready for data');
    });
```

There are other aspects of this project we didn't get the chance to cover. One of the bigger pieces is in Server.js, where the portal experience lives and lets you track the progress of all grid jobs in process. It includes a beautiful UI that's 100 percent Web-based. It's a live, constantly updating dashboard, complete with charts and graphs. We also didn't address the practical aspects of security and the threat of someone hijacking and modifying the JavaScript sent to the client and doing harm.

Wrapping Up

You could adapt all of this for other general-purpose grid computing problems. We think the more important take away from this article is the power and flexibility of Node.js. The repos on GitHub for Node.js exceed that of jQuery, a powerful testimony of how Node.js resonates with today's modern developer.

We'd like to thank the startup and partner evangelists, whose job it is to help companies and entrepreneurs understand and leverage the Microsoft stack and related technologies, many of which are open sourced. Warren Wilbee, West Region startup manager, seeded the Tech Crunch Disrupt team with some of his top players, including Felix Rieseberg, Helen Zeng, Steve Seow, Timothy Strimple and Will Tschumy. ■

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 Azure platform. You can read his blog at blogs.msdn.com/b/brunoterkaly.*

Ricardo Villalobos *is a seasoned software architect with more than 15 years of experience designing and creating applications for companies in multiple industries. Holding different technical certifications, as well as a master's degree in business administration from the University of Dallas, he works as a cloud architect in the DPE Globally Engaged Partners team for Microsoft, helping companies worldwide to implement solutions in Azure. You can read his blog at blog.ricardovillalobos.com.*

Terkaly and Villalobos jointly present at large industry conferences. They encourage readers of Azure Insider to contact them for availability. Terkaly can be reached at bterkaly@microsoft.com and Villalobos can be reached at Ricardo.Villalobos@microsoft.com.

Thanks to the following Microsoft technical experts for reviewing this article: *Gert Drapers, Cort Fritz and Tim Park*

Collaborating with Files?

View Sign Annotate Assemble Compare

GroupDocs.**Viewer**

Native-text, high-fidelity HTML5 document viewer with support for over 45 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 45 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.

document collaboration APIs

100% Standalone - No Office Automation

GROUPDOCS
Your Document Collaboration APIs

SALES INQUIRIES: +1 214 329 9760 sales@groupdocs.com www.groupdocs.com

JULIE LERMAN

Tips for Updating and Refactoring Your Entity Framework Code

I have worked with many clients to help refactor existing software that uses Entity Framework (EF). In the case of EF code, refactoring can have a variety of meanings and often involve an update. In this column, I'll take a look at some of the ways you might be overhauling your EF code, or applications that involve EF. For each of these approaches, I'll provide some guidance—based on my experiences working with clients' production applications—that should help you be well-prepared and avoid some headaches.

The changes I'll discuss are:

- Updating to a new version of Entity Framework
- Breaking up a large Entity Data Model
- Replacing the ObjectContext API with the DbContext API

I'll cover the first two at a high level this month and then follow up in my next column by digging into the last scenario with guidance, as well as some concrete examples.

Before you embark on any of these tasks, there's an initial bit of advice I highly recommend you follow: Do them one at a time. I've taken part in endeavors that include an old project that uses EF4, a huge model and the ObjectContext API. Attempting to change all three at the same time can only lead to tears and frustration—and, perhaps, worse. In this case, my suggested course was to first update the EF version without making any other changes, and then make sure everything continued to work. The next step involved identifying an area of the model that could be extracted into a new small model. But, initially, I let the new model continue to target the ObjectContext API. When everything was back in working order, the shift to the DbContext API was started, but for that small model only. Otherwise, too many things could break throughout the application and you'd be on a wild, incoherent mission to hunt down a variety of bugs. By shifting one small model at a time, you have a smaller surface area of broken code to rework and you can learn some good lessons and patterns that will make shifting the next small model much less painful.

Updating to a Newer Version of Entity Framework

Thanks to the EF team's focus on backward compatibility, moving from one version to another provides minimal friction. I'll focus on updating to EF6 —the major version of EF6, as well as its minor updates, such as EF6.02 and EF6.1.

The worst issues with moving from EF4, EF4.1 or EF5 to EF6 (which, in my opinion, really aren't so bad) result from some namespace changes. Because the original APIs are still in the Microsoft .NET Framework, having them duplicated in EF6 would cause a problem. So in the EF6 API, those classes are in a namespace that's different from System.Data to avoid conflicts. For example, there are a number of namespaces in the .NET-based EF APIs that begin with System.Data.Objects, System.Data.Common, System.Data.Mapping, System.Data.Query, as well as a few others. There are also some classes and enums that live directly in System.Data; for example, System.Data.EntityException and System.Data.EntityState. Most of the classes and namespaces that were tied directly to System.Data in this way were moved to the new namespace root, System.Data.Entity.Core. There are a handful of classes moved into System.Data.Entity, such as EntityState, which is now found at System.Data.Entity.EntityState. For example, the Mapping namespace is now System.Data.Entity.Core.Mapping while the Objects namespace is now System.Data.Entity.Core.Objects. See item 4 in the Data Developer Center documentation, "Upgrading to EF6" (bit.ly/OtrKvA), for the specific exceptions that didn't go into System.Data.Entity.Core.

When updating existing applications to EF6, I just let the compiler highlight the changes by showing me any "The type or namespace … is not found" errors and then I do some solution-wide finding and replacing to correct the namespaces.

As an example, I started with a small sample solution from the second edition of my book, "Programming Entity Framework." This solution was written using EF4, an EDMX model, code-generated POCO entity classes and the ObjectContext API. Code First and the DbContext API didn't exist at the time. Before I got started, I verified the application still worked (debugging in Visual Studio 2013).

Though I'm focusing on a bigger leap—from EF4 to EF6—you need to follow the same path of namespace fixes if you're going from EF5 to EF6, because these namespace changes occurred between EF5 and EF6. Moving from EF4 directly to EF6 has a few extra challenges, plus my old solution used a T4 template that doesn't have a direct replacement.

My Steps to Update from EF4 to EF6

If you're used to getting Entity Framework using the NuGet Package distribution, you'll need to think back to a time when EF was simply part of the .NET Framework and all of its DLLs lived in the Windows Global Assembly Cache (GAC). Before updating

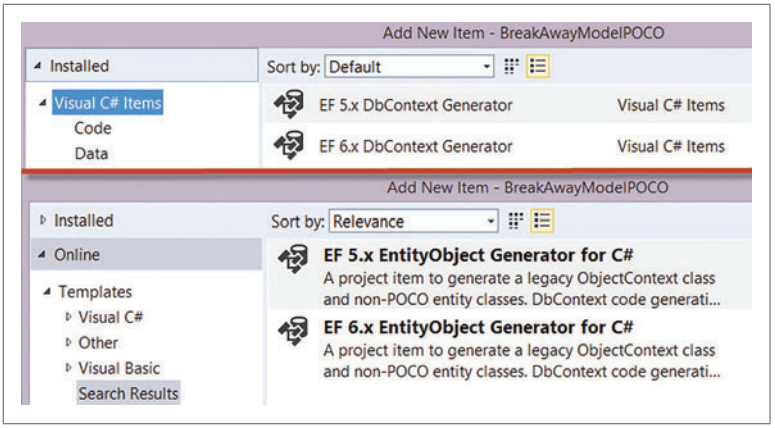


Figure 1 **You'll Find Templates to Generate DbContext Plus POCOs or ObjectContext Plus Non-POCOs.**

to EF6, I manually removed the references to System.Data.Entity (version 4.0.0.0) in each of my solution's projects. I also cleaned the solution (by right-clicking the solution in Solution Explorer and choosing Clean Solution) to be sure any of the original DLLs I may have forced into the BIN folders were gone. It turns out my diligence wasn't necessary because the NuGet package installer for EF6 removes the old references for you.

Then I used NuGet to install EF6 into the relevant projects and rebuilt the solution. The project dependencies in your own solutions will drive how quickly the namespace issues surface. With my solution, I was only shown one namespace error at first. I fixed that and rebuilt the solution, and then saw many more errors—all but one were namespace issues. For the one exception, the compiler provided a helpful message telling me that a less frequently used attribute I had taken advantage of (EdmFunction) had undergone a name change (and was therefore marked as Obsolete) and had been replaced by the attribute DbFunction.

After a series of iterative namespace fixes and rebuilds, which took only a few minutes for this small solution, I was able to successfully build and run the application—viewing, editing and saving data.

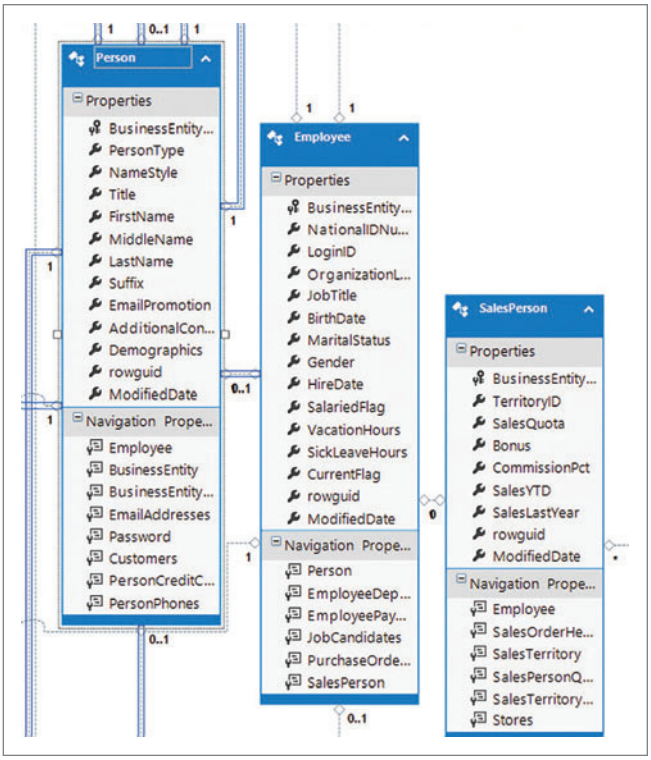


Figure 2 **SalesPerson and Territory Maintenance Can Be Extracted into a Separate Model with Little Impact to Other Entities**

Fixing the ObjectContext Plus POCO T4 Template

There's one other possible task to keep in mind. My solution used an EDMX—an Entity Data Model designed and maintained with the EF Designer. Because I created it in Visual Studio 2010 with EF4, it relied on an older code-generation template that generated an ObjectContext to manage all of the data persistence and caching. That template generated POCO classes (which have no dependency on Entity Framework) from the entities in the model. If I make any changes to my model, I'll need to regenerate the classes and the context. But that older template—the one that generates the context—isn't aware of the new namespaces. While there are DbContext templates and ObjectContext (plus EntityObjects) templates (see **Figure 1**), there's no replacement for my template that provides the ObjectContext plus POCOs combination. And to make things more interesting, I had customized the template I used. So rather than selecting a new template that wouldn't work with my application, I made two small changes to the Context.tt template that was stored in my existing solution:

1. On Line 42, "using System.Data.Objects;" becomes "using System.Data.Entity.Core.Objects;"
2. On line 43, "using System.Data.EntityClient;" becomes "using System.Data.Entity.Core.EntityClient;"

Now, any time I regenerate the classes from the model, the ObjectContext class will get the correct namespaces and my custom POCO-generated classes will continue to function in my application. Note that the Data Developer Center documentation I mentioned earlier explains how to use the supported templates.

Benefits You'll Gain Without Changing Any More Code

I found updating the application to use EF6 pretty painless. However, there's a very important point to consider. While the application is now using the most recent version of Entity Framework, it's benefitting only from the underlying improvements to Entity Framework—in particular some great performance gains that came in EF5 and EF6. Because the bulk of these performance gains came in EF5, moving from EF5 to EF6 without leveraging the other new features won't have as much impact. To see what else is new in EF6, check out my December 2013 article, "Entity Framework 6: The Ninja Edition" (bit.ly/1qJgwlf). Many of the improvements are related features of the DbContext API and Code First. If you want to update to EF6 and also update to DbContext, I recommend starting with the simple upgrade to EF6, and getting everything working

again before you begin to shift to the DbContext API. That's a more complex change that I'll cover in detail in my next column.

Even with these possibly minimal changes, your codebase is now ready to leverage the more modern APIs.

Breaking Up a Large Model

Whether you've used the EF Designer to create your model or the Code First workflow (see "Demystifying Entity Framework Strategies: Model Creation Workflow" at bit.ly/0u399J), models with many entities in them can cause design-time and even runtime problems. My personal experience is that large models are just too unwieldy and difficult to maintain. If you're using the Designer and have many entities (hundreds), not only does it take longer for the designer to open and display the model, it's difficult to visually navigate the model. Thankfully, the Designer gained some great capabilities in Visual Studio 2012 that help (see "Entity Framework Designer Gets Some Love in Visual Studio 2012" at bit.ly/1kV4vZ8).

Even so, I always recommend that models be smaller. In my column, "Shrink EF Models with DDD Bounded Contexts" (bit.ly/1isloGE), I talk about the benefits of smaller models, as well as some strategies for using them in your application. If you already have a big model, however, it can be a challenge to break that apart. I've had clients who were working with models they reverse-engineered from huge databases ending up with 700 or even 1,000 entities. Whether you're shrinking an EDMX model in the Designer or using Code First, the story is the same: Breaking up is hard to do.

Here are some useful pointers for breaking up a large model into smaller models for simpler maintenance and potentially better runtime performance.

Don't attempt to refactor the entire model at once. For each small model you extract, you'll need to do some additional code refactoring because references change and you might have some relationship code to tangle with as well.

So the first step is to identify a section of the model that's nearly autonomous. Don't worry about overlap at first. For example, you might be working on a system for a company that manufactures and sells products. The software may include a feature for maintaining your sales force—personnel data, contact information, sales territory and so forth. Another part of the software might reference those sales people when building a client's order based on the definition of your sales territories. And yet another part might be tracking their sales commissions. So tackle one particular scenario—say, maintaining the salesperson list—at a time.

Here are the steps for shifting to smaller models:

1. Identify the entities involved in that scenario (see **Figure 2**).
2. Create a completely new project.
3. In that project, define a new model (either with the EF Designer or using Code First) that's aware of the relevant entities.
4. Identify the application code involved with product maintenance.
5. Update the relevant code that uses the original context (for querying, saving changes or other functionality) so it uses the new context you've created.
6. Refactor the existing code until the target functionality is working. If you have automated testing in place, this might make the task of refactoring more efficient.

Some additional advice to keep in mind during this process:

- Do *not* remove these entities from the big model. Doing so would cause a lot of things to break. Just leave them there and forget about them for now.
- Keep a log of what kind of refactoring you had to do to get the application to work with the new smaller model. You'll learn patterns you can apply as you continue to break off other focused models from the big model.

By iteratively adding one new small model at a time to your solution and directing code to use it, your experience will be much more pleasant. In the big model, these same entities might be tangled up in relationships that have nothing to do with the SalesPerson maintenance task. For example, you might have a relationship from SalesPerson to a Sales Order, so you probably won't want to remove SalesPerson completely from the model. It's simpler to have a trimmed down, read-only SalesPerson type used as a reference during Order creation in one model and then the full-blown editable SalesPerson type (with no knowledge of Orders) used for maintenance in another model. When all is done, both entities can continue pointing to the same database. If you're using Code First and migrations, check the aforementioned article on shrinking EF models for guidance on sharing a database when you have multiple overlapping models.

Eventually you could end up with many small models and still some references to the big model. At that point, it will be easier to identify which entities are no longer touched in the big model and can be safely removed.

Patience: Yes, It's a Virtue

The most important takeaway is to approach these updates and refactors in small bites and at each iteration, get your tests and application functioning again before moving on. Consider updating to a newer version of Entity Framework as its own work item. And even that can be broken in two as you first concentrate on pulling in the newer APIs and making sure your code continues to run before changing the code to take advantage of new features. The same baby steps apply to breaking up a big model—especially if you're planning to update from ObjectContext to DbContext. Extract small models and refactor to make sure relevant logic can function with the new smaller model. Once that's working, it's time to break your ties to ObjectContext, which will break a bunch of code at first. But at least that broken code will be quarantined to a smaller area of your codebase and you'll be refactoring less code at a time.

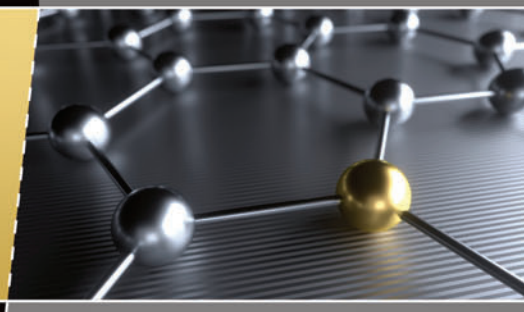
In my next column, I'll delve into the more painful but totally achievable goal of moving ObjectContext code to use the DbContext API. ■

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: *Rowan Miller*

Empower Your Customers

New! v5.0



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

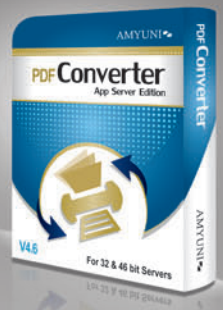
Complete Suite of Accurate PDF Components

- 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 certified PDF Converter printer driver
- Export PDF documents into other formats such as JPeg, PNG, XAML or HTML5

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 and Windows 8
- Standard PDF features included with a number of unique features. Interface with any .Net or ActiveX programming language
- Easy licensing and deployment to fit system administrator's requirements

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

A C# 6.0 Language Preview

Mark Michaelis

By the time you read this, Build—the Microsoft developer conference—will be over and developers will be thinking about how to respond to all that was presented: embrace immediately, watch with slight trepidation or ignore for the moment. For .NET/C# developers, the most significant announcement was undoubtedly the release of the next version of the C# compiler ("Roslyn") as open source. Related to that are the language improvements themselves. Even if you don't have plans to immediately adopt C# vNext—which I'll refer to henceforth and unofficially as C# 6.0—at a minimum, you should be aware of its features and take note of those that might make it worth jumping right in.

In this article, I'll delve into the details of what's available in C# 6.0 at the time of this writing (March 2014) or in the now open source bits downloadable from roslyn.codeplex.com. I'll refer to this as a single release that I'll term the March Preview. The C#-specific features of this March Preview are implemented entirely in the compiler, without any dependency on an updated Microsoft .NET Framework or runtime. This means you can adopt C# 6.0 in your development without having to upgrade the .NET Framework for either development or deployment. In fact, installing the C# 6.0 compiler from this release involves little more than installing a Visual Studio 2013 extension, which in turn updates the MSBuild target files.

As I introduce each C# 6.0 feature, you might want to consider the following:

- Was there a reasonable means of coding the same functionality in the past, such that the feature is mostly syntactic sugar—a short cut or streamlined approach? Exception filtering, for example, doesn't have a C# 5.0 equivalent, while primary constructors do.
- Is the feature available in the March Preview? Most features I'll describe are available, but some (such as a new binary literal) are not.
- Do you have any feedback for the team regarding the new language feature? The team is still relatively early in its release lifecycle and very interested in hearing your thoughts about the release (see msdn.com/Roslyn for feedback instructions).

Thinking through such questions can help you gauge the significance of the new features in relation to your own development efforts.

This article looks at a preview of C# 6.0. All information is subject to change.

This article discusses:

- Indexed members and element initializers
- Auto-properties with initializers
- Primary constructors
- Static using statements
- Declaration expressions
- Exception-handling improvements
- Additional numeric literal formats

Technologies discussed:

C#

Indexed Members and Element Initializers

To begin, consider the unit test in **Figure 1**.

Although it's somewhat obscured by the syntax, **Figure 1** is nothing more than a name-value collection. As such, the syntax

could be significantly cleaner: <index> = <value>. C# 6.0 makes this possible through the C# object initializers and a new index member syntax. The following shows int-based element initializers:

```
var cppHelloWorldProgram = new Dictionary<int, string>
{
  [10] = "main() {",
  [20] = "    printf(\"hello, world\")",
  [30] = "}"
};
Assert.AreEqual(3, cppHelloWorldProgram.Count);
```

Note that although this code uses an integer for the index, Dictionary<TKey,TValue> can support any type as an index (as long as it supports IComparable<T>). The next example presents a string for the index data type and uses an indexed member initializer to specify element values:

```
Dictionary<string, string> builtInDataTypes =
  new Dictionary<string, string> {
    ["Byte"] = "0 to 255",
    // ...
    // Error: mixing object initializers and
    // collection initializers is invalid
    // {" Boolean", "True or false."},
    ["Object"] = "An Object.",
    ["String"] = "A string of Unicode characters.",
    ["Decimal"] = "±1.0 × 10e?28 to ±7.9 × 10e28"
  };
```

Accompanying the new index member initialization is a new $ operator. This string indexed member syntax is specifically provided to address the prevalence of string-based indexing. With this new syntax, shown in **Figure 2**, it's possible to assign element values in syntax much more like in dynamic member invocation (introduced in C# 4.0) than the string notation used in the preceding example.

To understand the $ operator, take a look at the AreEqual function call. Notice the Dictionary member invocation of "$Boolean" on the builtInDataTypes variable—even though there's no "Boolean" member on Dictionary. Such an explicit member isn't required because the $ operator invokes the indexed member on the dictionary, the equivalent of calling buildInDataTypes["Boolean"].

As with any string-based operator, there's no compile-time verification that the string index element (for example, "Boolean") exists in the dictionary. As a result, any valid C# (case-sensitive) member name can appear after the $ operator.

To fully appreciate the syntax of indexed members, consider the predominance of string indexers in loosely typed data formats such as XML, JSON, CSV and even database lookups (assuming no Entity Framework code-generation magic). **Figure 3**, for example, demonstrates the convenience of the string indexed member using the Newtonsoft.Json framework.

One final point to note, just in case it's not already obvious, is that the $ operator syntax works only with indexes that are of type string (such as Dictionary<string, …>).

Auto-Properties with Initializers

Initializing a class today can be cumbersome at times. Consider, for example, the trivial case of a custom collection type (such as Queue<T>) that internally maintains a private System.Collections.Generic.List<T> property for a list of items. When instantiating the collection, you have to initialize the queue with the list of items it is to contain. However, the reasonable options for doing so with a property require a backing field along with an initializer or an else constructor, the combination of which virtually doubles the amount of required code.

Figure 1 **Assigning a Collection via a Collection Initializer (Added in C# 3.0)**

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using System.Collections.Generic;

// ...

[TestMethod]
public void DictionaryIndexWithoutDotDollar()
{
  Dictionary<string, string> builtInDataTypes = new Dictionary<string, string>()
  {
    {"Byte", "0 to 255"},
    // ...
    {"Boolean", "True or false."},
    {"Object", "An Object."},
    {"String", "A string of Unicode characters."},
    {"Decimal", "±1.0 × 10e-28 to ±7.9 × 10e28"}
  };

  Assert.AreEqual("True or false.", builtInDataTypes["Boolean"]);
}
```

Figure 2 **Initializing a Collection with an Indexed Member Assignment as Part of the Element Initializer**

```
[TestMethod]
public void DictionaryIndexWithDotDollar()
{
  Dictionary<string, string> builtInDataTypes = new Dictionary<string, string> {
    $Byte = "0 to 255",   // Using indexed members in element initializers
    // ...
    $Boolean = "True or false.",
    $Object = "An Object.",
    $String = "A string of Unicode characters.",
    $Decimal = "±1.0 × 10e?28 to ±7.9 × 10e28"
  };

  Assert.AreEqual("True or false.", builtInDataTypes.$Boolean);
}
```

Figure 3 **Leveraging the Indexed Method with JSON Data**

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using Newtonsoft.Json.Linq;

// ...

[TestMethod]
public void JsonWithDollarOperatorStringIndexers()
{
  // Additional data types eliminated for elucidation
  string jsonText = @"
    {
      'Byte':  {
        'Keyword':  'byte',
        'DotNetClassName':  'Byte',
        'Description':  'Unsigned integer',
        'Width':  '8',
        'Range':  '0 to 255'
                },
      'Boolean':  {
        'Keyword':  'bool',
        'DotNetClassName':  'Boolean',
        'Description':  'Logical Boolean type',
        'Width':  '8',
        'Range':  'True or false.'
                },

    }";

  JObject jObject = JObject.Parse(jsonText);

  Assert.AreEqual("bool", jObject.$Boolean.$Keyword);
}
```

Figure 4 **A Common Constructor Pattern**

```
[Serializable]
public class Patent
{
  public Patent(string title , string yearOfPublication)
  {
    Title = title;
    YearOfPublication = yearOfPublication;
  }
  public Patent(string title, string yearOfPublication,
    IEnumerable<string> inventors)
    : this(title, yearOfPublication)
  {
    Inventors = new List<string>();
    Inventors.AddRange(inventors);
  }

  [NonSerialized] // For example
  private string _Title;
  public string Title
  {
    get
    {
      return _Title;
    }
    set
    {
      if (value == null)
      {
        throw new ArgumentNullException("Title");
      }
      _Title = value;
    }
  }
  public string YearOfPublication { get; set; }

  public List<string> Inventors { get; private set; }

  public string GetFullName()
  {
    return string.Format("{0} ({1})", Title, YearOfPublication);
  }
}
```

With C# 6.0, there's a syntax shortcut: auto-property initializers. You can now assign to auto-properties directly, as shown here:

```
class Queue<T>
{
  private List<T> InternalCollection { get; } = new List<T>;

  // Queue Implementation
  // ...
}
```

Note that in this case, the property is read-only (no setter is defined). However, the property is still assignable at declaration time. A read/write property with a setter is also supported.

Primary Constructors

Along the same lines as property initializers, C# 6.0 provides syntactic shortcuts for the definition of a constructor. Consider the prevalence of the C# constructor and property validation shown in **Figure 4**.

There are several points to note from this common constructor pattern:

1. The fact that a property requires validation forces the underlying property field to be declared.
2. The constructor syntax is somewhat verbose with the all-too-common public class Patent{ public Patent(... repetitiveness.
3. "Title," in various versions of case sensitivity, appears seven times for a fairly trivial scenario—not including the validation.

Figure 5 **Using a Primary Constructor**

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

  private string _Title = title;
  public string Title
  {
    get
    {
      return _Title;
    }
    set
    {
      if (value == null)
      {
        throw new ArgumentNullException("Title");
      }
      _Title = value;
    }
  }

  public string YearOfPublication { get; set; } = yearOfPublication;

  public List<string> Inventors { get; } = new List<string>();

  public string GetFullName()
  {
    return string.Format("{0} ({1})", Title, YearOfPublication);
  }
}
```

4. The initialization of a property requires explicit reference to the property from within the constructor.

To remove some of the ceremony around this pattern, without losing the flavor of the language, C# 6.0 introduces property initializers and primary constructors, as shown in **Figure 5**.

In combination with property initializers, primary constructor syntax simplifies C# constructor syntax:

- Auto-properties, whether read-only (see the Inventors property with only a getter) or read-write, (see the YearOfPublication property with both a setter and a getter), support property initialization such that the initial value of the property can be assigned as part of the property declaration. The syntax matches what's used when assigning fields a default value at declaration time (declaration assigned _Title, for example).
- By default, primary constructor parameters aren't accessible outside of an initializer. For example, there's no yearOfPublication field declared on the class.
- When leveraging property initializers on read-only properties (getter only), there's no way to provide validation. (This is due to the fact that in the underlying IL implementation, the primary constructor parameter is assigned to the backing field. Also noteworthy is the fact that the backing field will be defined as read-only in the IL if the auto-property has only a getter.)
- If specified, the primary constructor will (and must) always execute last in the constructor chain (therefore, it can't have a this(...) initializer).

Built for Today... Ready for Tomorrow

Touch-Enabled .NET Controls by DevExpress.

Create solutions your customers expect today and leverage your existing skillsets to build next generation applications for tomorrow. DevExpress Controls are built to emulate the UI experiences at the heart of Microsoft Office AND to enable touch-first experiences for next-generation devices like iPad and Surface.

Your next great app starts here.
DevExpress.com/Touch

DevExpress®

Figure 6 **Simplifying Code Clutter with Using Static**

```
using System;
using System.Console;

public class Program
{
  private static void Main()
  {
    ConsoleColor textColor = ForegroundColor;
    try
    {
      ForegroundColor = ConsoleColor.Red;
      WriteLine("Hello, my name is Inigo Montoya... Who are you?: ");
      ForegroundColor = ConsoleColor.Green;
      string name = ReadLine(); // Respoond: No one of consequence
      ForegroundColor = ConsoleColor.Red;
      WriteLine("I must know.");
      ForegroundColor = ConsoleColor.Green;
      WriteLine("Get used to disappointment");
    }
    finally
    {
      ForegroundColor = textColor;
    }
  }
}
```

For another example, consider the declaration of a struct, which guidelines indicate should be immutable. The following shows a property-based implementation (versus the atypical public field approach):

```
struct Pair(string first, string second, string name)
{
  public Pair(string first, string second) : this(first, second, first+"-"+second)
  {
  }

  public string First { get; } = second;
  public string Second { get; } = first;
  public string Name { get; } = name;

  // Possible equality implementation
  // ...
}
```

Note that in the implementation of Pair, there's a second constructor that invokes the primary constructor. In general, all struct constructors must—either directly or indirectly—invoke the primary constructor via a call to the this(...) initializer. In other words, it isn't necessary that all constructors call the primary constructor directly, but that at the end of the constructor chain the primary constructor is called. This is necessary because it's the primary constructor that calls the base constructor initializer and doing so provides a little protection against some common initialization mistakes. (Note that, as was true in C# 1.0, it's still possible to instantiate a struct without invoking a constructor. This, for example, is what happens when an array of the struct is instantiated.)

Figure 7 **Declaration Expression Examples**

```
public string FormatMessage(string attributeName)
{
  string result;
  if(! Enum.TryParse<FileAttributes>(attributeName, out var attributeValue) )
  {
    result = string.Format(
      "'{0}' is not one of the possible {2} option combinations ({1})",
      attributeName, string.Join(",", string[] fileAtrributeNames =
      Enum.GetNames(typeof (FileAttributes))),
      fileAtrributeNames.Length);
  }
  else
  {
    result = string.Format("'{0}' has a corresponding value of {1}",
      attributeName, attributeValue);
  }
  return result;
}
```

Whether the primary constructor is on a custom struct or class data type, the call to the base constructor is either implicit (therefore invoking the base class's default constructor) or explicit, by calling a specific base class constructor. In the latter case, for a custom exception to invoke a specific System.Exception constructor, the target constructor is specified after the primary constructor:

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

One detail to be aware of regarding primary constructors relates to avoiding duplicate, potentially incompatible, primary constructors on partial classes: Given multiple parts of a partial class, only one class declaration can define the primary constructor and, similarly, only this primary constructor can specify the base constructor invocation.

There's one significant caveat to consider in regard to primary constructors as they're implemented in this March Preview: There's no way to provide validation to any of the primary constructor parameters. And, because property initializers are only valid for auto-properties, there's no way to implement validation in the property implementation, either, which potentially exposes public property setters to the assignment of invalid data post instantiation, as well. The obvious workaround for the moment is to not use the primary constructor feature when validation is important.

Although somewhat tentative at the moment, there's a related feature called the field parameter under consideration. The inclusion of an access modifier in the primary constructor parameter (such as private string title) will cause the parameter to be captured into class scope as a field with the name of title—matching the name and casing of the parameter). As such, title is available from within the Title property or any other instance class member. Furthermore, the access modifier allows the entire field syntax to be specified—including additional modifiers such as readonly, or even attributes like these:

```
public class Person(
  [field: NonSerialized] private string firstName, string lastName)
```

Note that without the access modifier, other modifiers (including attributes) aren't allowed. It's the access modifier that indicates the field declaration is to occur in-line with the primary constructor.

(The bits available to me at the time of this writing didn't include the field parameter implementation, but I am assured by the language team they will be included in the Microsoft Build version, so you should be able to try out field parameters by the time you read this. Given the relative "freshness" of this feature, however, don't hesitate to provide feedback at msnd.com/Roslyn so it can be considered before the process is too far along for changes.)

Static Using Statements

Another C# 6.0 "syntactic sugar" feature is the introduction of using static. With this feature, it's possible to eliminate an explicit reference to the type when invoking a static method. Furthermore,

using static lets you introduce only the extension methods on a specific class, rather than all extension methods within a namespace. **Figure 6** provides a "Hello World" example of using static on System.Console.

In this example, the Console qualifier was dropped a total of nine times. Admittedly, the example is contrived, but even so, the point is clear. Frequently a type prefix on a static member (including properties) doesn't add significant value and eliminating it results in code that's easier to write and read.

Although not working in the March Preview, a second (planned) feature of using static is under discussion. This feature is support for importing only extension methods of a specific type. Consider, for example, a utility namespace that includes numerous static types with extension methods. Without using static, all (or no) extension methods in that namespace are imported. With using static, however, it's possible to pinpoint the available extension methods to a specific type—not to the more general namespace. As a result, you could call a LINQ standard query operator by just specifying using System.Linq.Enumerable; instead of the entire System.Linq namespace.

Unfortunately, this advantage isn't always available (at least in the March Preview) because only static types support using static, which is why, for example, there's no using System.ConsoleColor statement in **Figure 6**. Given the current preview nature of C# 6.0, whether the restriction will remain is still under review. What do you think?

Declaration Expressions

It's not uncommon that in the midst of writing a statement, you find you need to declare a variable specifically for that statement. Consider two examples:

- When coding an int.TryParse statement, you realize you need to have a variable declared for the out argument into which the parse results will be stored.
- While writing a for statement, you discover the need to cache a collection (such as a LINQ query result) to avoid re-executing the query multiple times. In order to achieve this, you interrupt the thought process of writing the for statement to declare a variable.

To address these and similar annoyances, C# 6.0 introduces declaration expressions. This means you don't have to limit variable declarations to statements only, but can use them within expressions, as well. **Figure 7** provides two examples.

Figure 8 **Await Calls from Within a Catch Block**

```
try
{
  WebRequest webRequest =
    WebRequest.Create("http://IntelliTect.com");

  WebResponse response =
    await webRequest.GetResponseAsync();

  // ...
}
catch (WebException exception)
{
  await WriteErrorToLog(exception);
}
```

In the first highlight in **Figure 7**, the attributeValue variable is declared in-line with the call to Enum.TryParse rather than in a separate declaration beforehand. Similarly, the declaration of fileAttributeNames appears on the fly in the call to string.Join. This enables access to the Length later in the same statement. (Note that the fileAttributeNames.Length is substitution parameter {2} in the string.Format call, even though it appears earlier in the format string—thus enabling fileAttributeNames to be declared prior to accessing it.)

The scope of a declaration expression is loosely defined as the scope of the statement in which the expression appears. In **Figure 7**, the scope of attributeValue is that of the if-else statement, making it accessible both in the true and false blocks of the conditional. Similarly, fileAttributeNames is available only in the first half of the if-statement, the portion matching the scope of the string.Format statement invocation.

> Frequently a type prefix on a static member doesn't add significant value and eliminating it results in code that's easier to write and read.

Wherever possible the compiler will enable the use of implicitly typed variables (var) for the declaration, inferring the data type from the initializer (declaration assignment). However, in the case of out arguments, the signature of the call target can be used to support implicitly typed variables even if there's no initializer. Still, inference isn't always possible and, furthermore, it may not be the best choice from a readability perspective. In the TryParse case in **Figure 7**, for example, var works only because the type argument (FileAttributes) is specified. Without it, a var declaration wouldn't compile and instead the explicit data type would be required:

```
Enum.TryParse(attributeName, out FileAttributes attributeValue)
```

In the second declaration expression example in **Figure 7**, an explicit declaration of string[] appears to identify the data type as an array (rather than a List<string>, for example). The guideline is standard to the general use of var: Consider avoiding implicitly typed variables when the resulting data type isn't obvious.

The declaration expression examples in **Figure 7** could all be coded by simply declaring the variables in a statement prior to their assignment.

Exception-Handling Improvements

There are two new exception-handling features in C# 6.0. The first is an improvement in the async and await syntax and the second is support for exception filtering.

When C# 5.0 introduced the async and await (contextual) keywords, developers gained a relatively easy way to code the Task-based Asynchronous Pattern (TAP) in which the compiler takes on the laborious and complex work of transforming C# code into

an underlying series of task continuations. Unfortunately, the team wasn't able to include support for using await from within catch and finally blocks in that release. As it turned out, the need for such an invocation was even more common than initially expected. Thus, C# 5.0 coders had to apply significant workarounds (such as leveraging the awaiter pattern). C# 6.0 does away with this deficiency, and now allows await calls within both catch and finally blocks (they were already supported in try blocks), as shown in **Figure 8**.

The other exception improvement in C# 6.0—support for exception filters—brings the language up-to-date with other .NET languages, namely Visual Basic .NET and F#. **Figure 9** shows the details of this feature.

Notice the additional if expression that follows the catch expression. The catch block now verifies that not only is the exception of type Win32Exception (or derives from it), but also verifies additional conditions—the particular value of the error code in this example. In the unit test in **Figure 9**, the expectation is the catch block will not catch the exception—even though the exception type matches—instead, the exception will escape and be handled by the ExpectedException attribute on the test method.

Note that unlike some of the other C# 6.0 features discussed earlier (such as the primary constructor), there was no equivalent alternate way of coding exception filters prior to C# 6.0. Until now, the only approach was to catch all exceptions of a particular type, explicitly check the exception context, and then re-throw the exception if the current state wasn't a valid exception-catching scenario. In other words, exception filtering in C# 6.0 provides functionality that hitherto wasn't equivalently possible in C#.

Figure 9 **Leveraging Exception Filters to Pinpoint Which Exception to Catch**

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using System.ComponentModel;
using System.Runtime.InteropServices;

// ...

[TestMethod][ExpectedException(typeof(Win32Exception))]
public void ExceptionFilter_DontCatchAsNativeErrorCodeIsNot42()
{
  try
  {
    throw new Win32Exception(Marshal.GetLastWin32Error());
  }
  catch (Win32Exception exception) if (exception.NativeErrorCode == 0x00042)
  {
    // Only provided for elucidation (not required).
    Assert.Fail("No catch expected.");
  }
}
```

Figure 10 **Assigning Binary Literals for Enum Values**

```
[Serializable][Flags]
[System.Runtime.InteropServices.ComVisible(true)]
public enum FileAttributes
{
  ReadOnly =          0b00_00_00_00_00_00_01, // 0x0001
  Hidden =            0b00_00_00_00_00_00_10, // 0x0002
  System =            0b00_00_00_00_00_01_00, // 0x0004
  Directory =         0b00_00_00_00_00_10_00, // 0x0010
  Archive =           0b00_00_00_00_01_00_00, // 0x0020
  Device =            0b00_00_00_00_10_00_00, // 0x0040
  Normal =            0b00_00_00_01_00_00_00, // 0x0080
  Temporary =         0b00_00_00_10_00_00_00, // 0x0100
  SparseFile =        0b00_00_01_00_00_00_00, // 0x0200
  ReparsePoint =      0b00_00_10_00_00_00_00, // 0x0400
  Compressed =        0b00_01_00_00_00_00_00, // 0x0800
  Offline =           0b00_10_00_00_00_00_00, // 0x1000
  NotContentIndexed = 0b01_00_00_00_00_00_00, // 0x2000
  Encrypted =         0b10_00_00_00_00_00_00  // 0x4000
}
```

Additional Numeric Literal Formats

Though it's not yet implemented in the March Preview, C# 6.0 will introduce a digit separator, the underscore (_), as a means of separating the digits in a numerical literal (decimal, hex or binary). The digits can be broken into whatever grouping makes sense for your scenario. For example, the maximum value of an integer could be grouped into thousands:

```
int number = 2_147_483_647;
```

The result makes it clearer to see the magnitude of a number, whether decimal, hex or binary.

The digit separator is likely to be especially helpful for the new C# 6.0 numeric binary literal. Although not needed in every program, the availability of a binary literal could improve maintainability when working with binary logic or flag-based enums. Consider, for example, the FileAttribute enum shown in **Figure 10**.

Now, with binary numeric literals, you can show more clearly which flags are set and not set. This replaces the hex notation shown in comments or the compile time calculated shift approach:

```
Encrypted = 1<<14.
```

(Developers eager to try this feature immediately can do so in Visual Basic .NET with the March Preview release.)

Wrapping Up

In considering only these language changes, you'll notice there's nothing particularly revolutionary or earth-shattering in C# 6.0. If you compare it to other significant releases, like generics in C# 2.0, LINQ in C# 3.0 or TAP in C# 5.0, C# 6.0 is more of a "dot" release than a major one. (The big news being the compiler has been released as open source.) But just because it doesn't revolutionize your C# coding doesn't mean it hasn't made real progress in eliminating some coding annoyances and inefficiencies that, once in your quiver of everyday use, you'll quickly take for granted. The features that rank among my particular favorites are the $ operator (string index members), primary constructors (without field parameters), using static and declaration expressions. I expect each of these to quickly become the default in my coding, and likely even added into coding standards in some cases. ■

Mark Michaelis *is the founder of IntelliTect and serves as the chief technical architect and trainer. Since 1996, he's 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 and has written numerous articles and books, and is currently working on the next edition of "Essential C#" (Addison-Wesley Professional). Michaelis holds a Bachelor of Arts in Philosophy from the University of Illinois and a Master of Computer Science from the Illinois Institute of Technology. When not bonding with his computer, he's busy with his family or training for another Ironman.*

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

Next-Generation Development with Application Insights

Charles Sterling

The pace of software delivery has increased dramatically over the past few years. Developers have evolved from waterfall to agile to today's continuous release cadence. In doing so, the need has intensified for better, faster and more direct feedback. Responsiveness is the name of the game. Decision makers need tools that provide integrated analytics, and make timely data instantly available to the entire team.

The new Microsoft Application Insights, announced at the Visual Studio 2013 launch, is a suite of services designed around the key questions high-performance delivery teams need answered: Is our application available? Is it performing? Are we delivering the features our users want?

Application Insights isn't limited to operations. To eliminate handoffs and speed up the flow of information throughout the team, it integrates with tools developers and operations already use—Visual Studio and Visual Studio Online. This makes it easy for all team members to get the information they need.

This article discusses:

- Development monitoring with Application Insights
- How to monitor older Visual Studio and other apps
- Integrating app performance and troubleshooting data

Technologies discussed:

Visual Studio, JavaScript

Application Insights is designed to work with services built into the Microsoft .NET Framework, Java, Windows Azure services, Web sites, Windows Store applications and Windows Phone 8 applications. With complete end-to-end monitoring, you get a true 360-degree picture of your application, not just small pieces of isolated data.

Getting Started with Application Insights

Getting started with Application Insights is simple. To add the Application Insights telemetry to Web, Windows Phone or Windows Store applications, download the Application Insights Tools for Visual Studio extension, which you'll find in the Visual Studio Gallery (aka.ms/aivsix). Future versions of Visual Studio won't require this extra step.

For new Projects in Visual Studio 2013, select Add Application Insights to Project when you create a new Project (see **Figure 1**).

To use Application Insights with existing applications, right-click on the project and choose Add Applications Insights Telemetry to Project (see **Figure 2**).

After adding Application Insights, your project will have three new nodes with shortcuts to Availability Monitoring, Performance Monitoring and Usage Analytics data in Visual Studio Online (see **Figure 3**).

Implement Usage Monitoring

Once you've added Application Insights to a new or existing project, usage monitoring on your Web, Windows Store, and Windows Phone

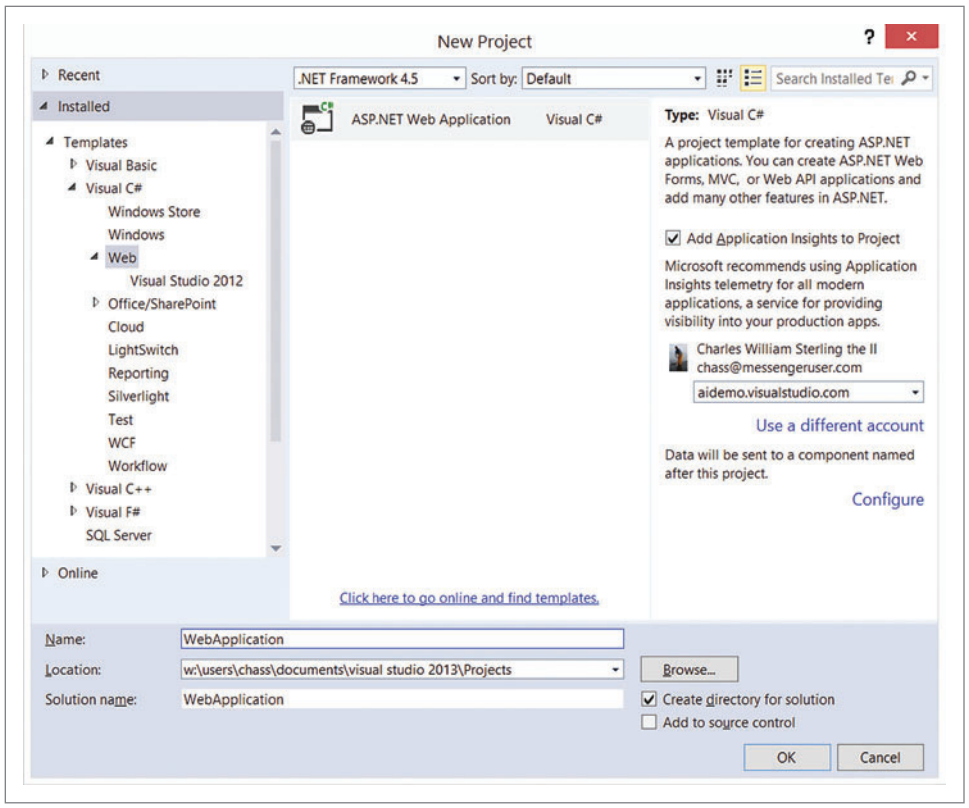


Figure 1 **Add Application Insights to New Visual Studio 2013 Projects**

applications is automatically enabled. For older and non-Visual Studio Web applications, you can add this same functionality by inserting a block of JavaScript into your application. You can get to this JavaScript block by clicking Add Application (see **Figure 4**)

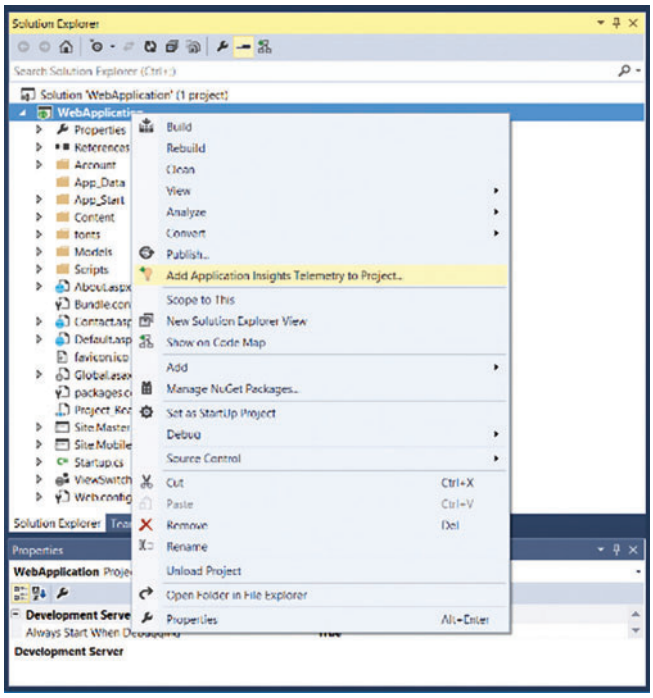


Figure 2 **Right-Click to Add Application Insights to an Existing Project**

or by going to the Control Panel and selecting Get configuration keys and downloads.

Implement Performance Monitoring

Despite the name, Performance Monitoring gives you a ton of information beyond just performance data. It will notify you of exceptions, call stack information, dependency information, object allocation and even information about the underlying infrastructure. Microsoft Monitoring Agent (MMA) also automatically collects the IntelliTrace logs for exceptions and slow calls in your code. In most cases, you can activate performance monitoring by simply installing MMA, which you'll find at aka.ms/aimma.

When you install MMA, it will default to monitoring all Web applications on your machine. This is probably fine for a development box, but might not be ideal for production servers with many Web applications. MMA shouldn't cause any more than a 5 percent performance degradation while monitoring your application.

To enable MMA on applications you add later, you can easily manually activate monitoring via a Windows PowerShell command:

```
Start-WebApplicationMonitoring -name "www.microsoft.com/games" -mode
Monitor -outputchannel cloud
```

In future releases of MMA and Visual Studio, you won't need to take this step.

You can also activate performance monitoring on Java and Windows Azure applications. The simplest way to get started is

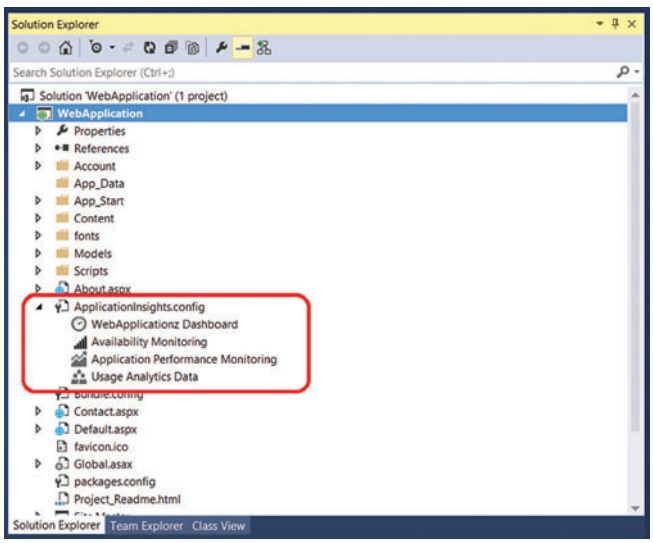


Figure 3 **New Nodes Appear in the Enabled Project**

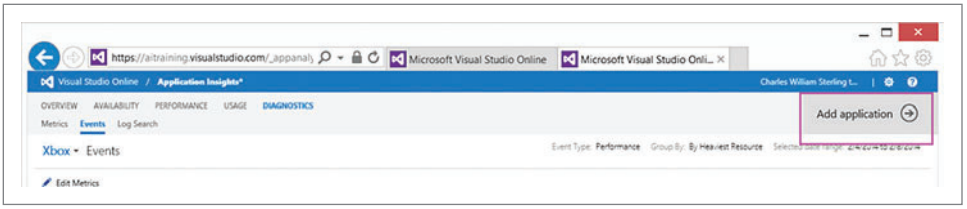


Figure 4 **Select Add Application to Add JavaScript Block to Older Apps**

to click Add Application in the Application Insights portal, as I mentioned earlier for implementing Usage Monitoring.

Implement Availability Monitoring

Availability Monitoring works for any Web application, regardless of the platform on which it's running. It only has to be accessible via the Internet. You can test the availability and performance of your Web application from across the world. This module is also the easiest to get working.

When you select the Availability menu within Application Insights, you'll be prompted to supply the URL for your Web application. This will create a simple URL-based synthetic monitor with a single monitoring location.

If you need to monitor more sophisticated transactions, you'll probably be better off using a Visual Studio coded Web performance test. The synthetic monitor is based on the same recording functionality commonly used for load testing Web applications. This lets you test a complex set of actions. To create a single URL or multi-step synthetic monitor, click on the green Add new synthetic monitor icon and configure the desired settings (see **Figure 5**).

Put Application Insights to Work

Over the last year, nearly 100 internal Microsoft teams and external industry experts tried out early versions of Application Insights and provided feedback as part of a formal Technical Adoption Program. Some of what they uncovered was surprising to the development team, especially the strong interest and engagement by product owners and non-technical team members.

The early adopters agreed a key value of Application Insights is the ability to speed up the development cycle by bringing all this analytics information into a single location in Visual Studio Online.

Measure a Web Campaign One of the first external customers to look at Application Insights was Wintellect—a training and consulting firm. They wanted to understand the impact of course descriptions for their new on-demand training product, WintellectNOW.

Using the Page Views report in Application Insights, Wintellect developers were able to add a JavaScript function to the Sign Up Now button, similar to this:

```
function trackCourse()
{
  window.__da.trackEvent("Course",
                         window.location.hostname + window.location.pathname,
                         {"CourseID": "Test401"},
                         {"RatingValue": 400});
}
```

This let them measure and visualize which course descriptions were most effectively driving actual registrations. For more information on implementing custom events in Application Insights, see aka.ms/aijs.

Measure Global Web Traffic Wintellect was scheduled to exhibit at TechEd 2013 Europe in Madrid, Spain. The business staff wanted an easy way to measure whether the company's presence would increase awareness of its offerings in the European market.

The company set up usage reports using Application Insights, and compared the week before TechEd to the week after. Traffic from Europe increased by 7 percent, and it doubled its traffic from Spain. Wintellect didn't need any special effort by developers to measure these results, so its technical teams were able to stay focused on deliverables.

Streamline Find, Fix and Release

Application Insights is at work at Microsoft as well. The service engineers in charge of the main Microsoft Web site and homepage manage more than 400 applications on a day-to-day basis. Their highest priority is to reduce the time from discovering an issue to getting a fix in place. By setting up dashboards and alerts with Application Insights, they get real-time notification of failing availability tests, performance events or a degrading performance index. This helps the engineers resolve issues before customers even notice anything is wrong.

One team set up an availability monitor with a maximum allowable runtime, with an alert to indicate any time the threshold is crossed. The engineers can then ascertain the cause of failure directly from within the Web view, or download it to Visual Studio and view it as a Web Test Result. The Synthetic Monitors report indicates these tests

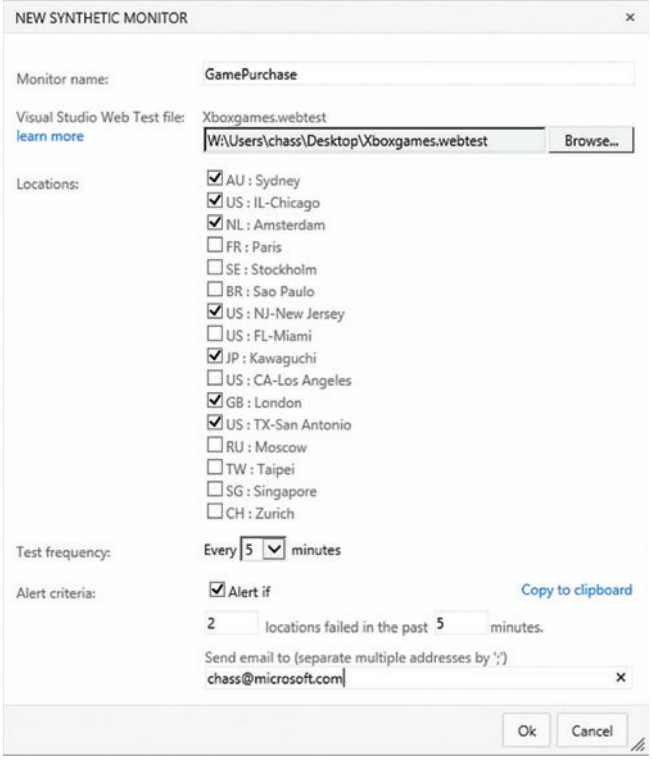


Figure 5 **Configure the Settings for a New Synthetic Monitor**

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, 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
- Native WinRT Libraries for Windows Store & Zero-Footprint HTML5/JavaScript Controls

BEST SELLER

ComponentOne Studio Enterprise 2014 v1 | 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
- Visual Studio 2013 and Bootstrap support
- Advanced theming tools for WinForms and ASP.NET
- 40+ UI widgets built with HTML5, jQuery, CSS3, and SVG
- Windows Store Sparkline, DropDown, & Excel controls

BEST SELLER

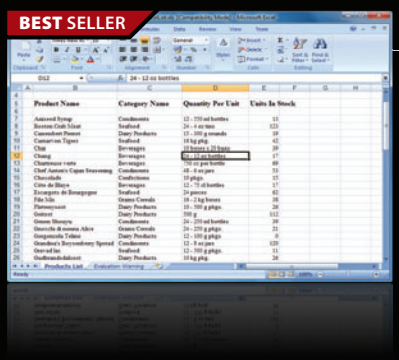
Help & Manual Professional | from $583.10

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 PDF files

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

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

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

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

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

GSA Schedule Contract GS-35F-0188R

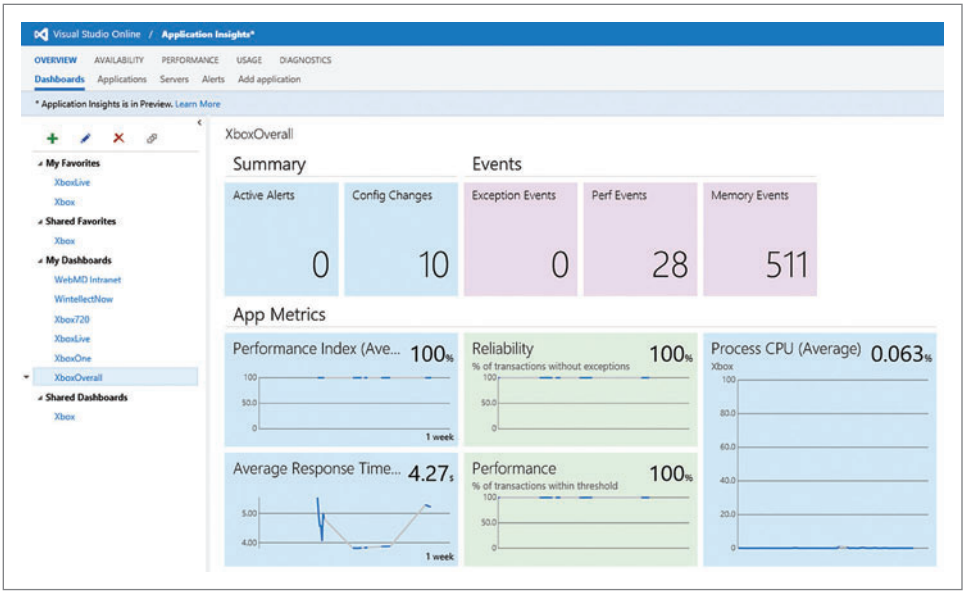


Figure 6 **The Application Insights Dashboard**

only started failing after making a deployment. Then it continued running successfully after another deployment. Around four hours later, 11 configuration changes were made. They were able to tie the availability issue directly back to actual code and configuration changes. This helped them diagnose the root cause of the event right away.

With Application Insights, you can optimize your applications before they even generate an alert. In the dashboard, there are Active Alerts, Exception Events, Performance Events, Memory Events, and Performance and Reliability graphs. All these metrics provide easily understandable information for an engineering team looking to improve its applications.

Selecting any of these tiles will take you to the data most likely to lead to an action. For example, clicking on the Performance graph in the Application Insights Dashboard (**Figure 6**) will take you to the Performance page (**Figure 7**). In this example, you can see there's a strong correlation between dependencies, a Web service and response times.

Click on the Memory, Exception or Performance Events tiles in the Dashboard to get to the Events page. Here, you can filter, select, open the memory diagnostics session, start an IntelliTrace debug session or view the changeset that caused the event in Visual Studio.

Wrapping Up

These are just a few examples of how development teams are using Application Insights to work more closely with their operations teams to more quickly deliver better software. You can access Application Insights through Visual Studio Online (visualstudio.com).

In a future article, I'll cover integrating cloud-based load testing with Application Insights. For more on creating Web tests, go to bit.ly/1im10Yl. For more about Availability Monitoring with Application Insights, go to bit.ly/1gxgLYk.

With the ease of adding monitoring to your code, the close integration with Visual Studio Online, and the time savings, you'll definitely want to check out these scenarios and realize what you can accomplish with Application Insights. ■

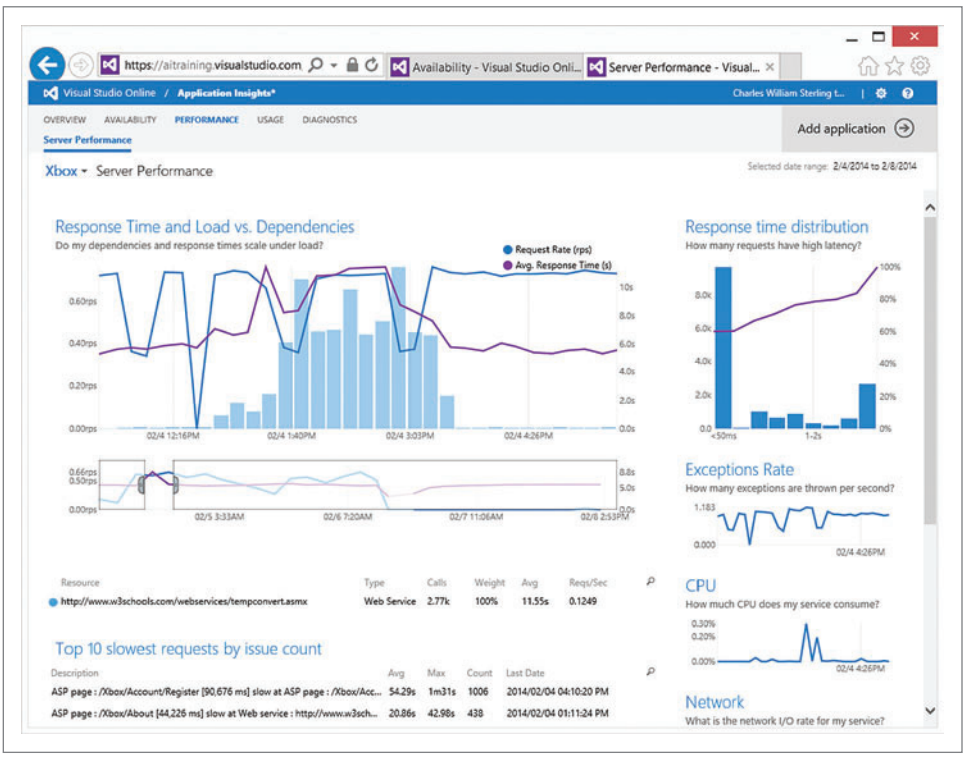


Figure 7 **The Application Insights Performance Page**

Charles Sterling *has been at Microsoft for 20 years. Before Microsoft he was a marine biologist working for the United States National Marine Fisheries doing marine mammal research on the Bering Sea. After a six-year tour in Australia as a Microsoft product manager, Sterling is currently back in Redmond as a senior program manager on the Visual Studio development team. Reach him at chass@microsoft.com.*

Thanks to the following technical experts for reviewing this article: *Cheryl Hammond (Northwest Cadence) and John Robbins (Wintellect)*

Patterns for Asynchronous MVVM Applications: Services

Stephen Cleary

This is the third article in a series on combining async and await with the established Model-View-ViewModel (MVVM) pattern. In the first article, I developed a method for data binding to an asynchronous operation. In the second, I considered a few possible implementations of an asynchronous ICommand. Now, I'll turn to the service layer and address asynchronous services.

I'm not going to deal with a UI at all. In fact, the patterns in this article aren't specific to MVVM; they pertain equally well to any type of application. The asynchronous data binding and command patterns explored in my earlier articles are quite new; the asynchronous service patterns in this article are more established. Still, even established patterns are just patterns.

Asynchronous Interfaces

"Program to an interface, not an implementation." As this quote from "Design Patterns: Elements of Reusable Object-Oriented Software" (Addison-Wesley, 1994, p. 18) suggests, interfaces are a critical component of proper object-oriented design. They allow your code to use an abstraction rather than a concrete type, and they give your code a "junction point" you can splice into for unit testing. But is it possible to create an interface with asynchronous methods?

The answer is yes. The following code defines an interface with an asynchronous method:

```
public interface IMyService
{
  Task<int> DownloadAndCountBytesAsync(string url);
}
```

The service implementation is straightforward:

```
public sealed class MyService : IMyService
{
  public async Task<int> DownloadAndCountBytesAsync(string url)
  {
    await Task.Delay(TimeSpan.FromSeconds(3)).ConfigureAwait(false);
    using (var client = new HttpClient())
    {
      var data = await client.GetByteArrayAsync(url).ConfigureAwait(false);
      return data.Length;
    }
  }
}
```

Figure 1 shows how the code that consumes the service calls the asynchronous method defined on the interface.

This may seem like an overly simplistic example, but it illustrates some important lessons about asynchronous methods.

The first lesson is: Methods are not awaitable, types are. It is the type of an expression that determines whether that expression is awaitable. In particular, UseMyService.IsLargePageAsync awaits the result of IMyService.DownloadAndCountBytesAsync.

This article discusses:

- Asynchronous interfaces
- Asynchronous unit testing
- Asynchronous factories
- Asynchronous resources
- The asynchronous initialization pattern

Technologies discussed:

Asynchronous Programming, Model-View-ViewModel Pattern

Figure 1 **UseMyService.cs: Calling the Async Method Defined on the Interface**

```
public sealed class UseMyService
{
  private readonly IMyService _service;

  public UseMyService(IMyService service)
  {
    _service = service;
  }

  public async Task<bool> IsLargePageAsync(string url)
  {
    var byteCount = await _service.DownloadAndCountBytesAsync(url);
    return byteCount > 1024;
  }
}
```

The interface method is not (and cannot be) marked async. IsLargePageAsync can use await because the interface method returns a Task, and tasks are awaitable.

The second lesson is: Async is an implementation detail. UseMyService neither knows nor cares whether the interface methods are implemented using async or not. The consuming code cares only that the method returns a task. Using async and await is a common way to implement a task-returning method, but it's not the only way. For example, the code in **Figure 2** uses a common pattern for overloading asynchronous methods.

Note that one overload merely calls the other and returns its task directly. It's possible to write that overload using async and await, but that would only add overhead and provide no benefit.

Asynchronous Unit Testing

There are other options for implementing task-returning methods. Task.FromResult is a common choice for unit test stubs, because it's the easiest way to create a completed task. The following code defines a stub implementation of the service:

```
class MyServiceStub : IMyService
{
  public int DownloadAndCountBytesAsyncResult { get; set; }

  public Task<int> DownloadAndCountBytesAsync(string url)
  {
    return Task.FromResult(DownloadAndCountBytesAsyncResult);
  }
}
```

You can use this stub implementation to test UseMyService, as shown in **Figure 3**.

This example code uses MSTest, but most other modern unit test frameworks also support asynchronous unit tests. Just make sure your unit tests return task; avoid async void unit test methods. Most unit test frameworks do not support async void unit test methods.

Figure 2 **AsyncOverloadExample.cs: Using a Common Pattern for Overloading Async Methods**

```
class AsyncOverloadExample
{
  public async Task<int> RetrieveAnswerAsync(CancellationToken cancellationToken)
  {
    await Task.Delay(TimeSpan.FromSeconds(3), cancellationToken);
    return 42;
  }

  public Task<int> RetrieveAnswerAsync()
  {
    return RetrieveAnswerAsync(CancellationToken.None);
  }
}
```

When unit testing synchronous methods, it's important to test how the code behaves both in success and failure conditions. Asynchronous methods add a wrinkle: It's possible for an asynchronous service to succeed or throw an exception, synchronously or asynchronously. You can test all four of these combinations if you want, but I find it's usually sufficient to test at least asynchronous success and asynchronous failure, plus synchronous success if necessary. The synchronous success test is useful because the await operator will act differently if its operation has already completed. However, I don't find the synchronous failure test as useful, because failure isn't immediate with most asynchronous operations.

As of this writing, some popular mocking and stubbing frameworks will return default(T) unless you modify that behavior. The default mocking behavior doesn't work well with asynchronous methods because asynchronous methods should never return a null task (according to the Task-based Asynchronous Pattern, which you'll find at bit.ly/1ifhkK2). The proper default behavior would be to return Task.FromResult(default(T)). This is a common problem when unit testing asynchronous code; if you're seeing unexpected NullReferenceExceptions in your tests, ensure the mock types are implementing all the task-returning methods. I hope that mocking and stubbing frameworks will become more async-aware in the future, and implement better default behavior for asynchronous methods.

Asynchronous Factories

The patterns so far have demonstrated how to define an interface with an asynchronous method; how to implement it in a service; and how to define a stub for testing purposes. These are sufficient for most asynchronous services, but there's a whole other level of complexity that applies when a service implementation must do some asynchronous work before it's ready to be used. Let me describe how to handle the situation where you need an asynchronous constructor.

Figure 3 **UseMyServiceUnitTests.cs: Stub Implementation to Test UseMyService**

```
[TestClass]
public class UseMyServiceUnitTests
{
  [TestMethod]
  public async Task UrlCount1024_IsSmall()
  {
    IMyService service = new MyServiceStub {
DownloadAndCountBytesAsyncResult = 1024 };
    var logic = new UseMyService(service);
    var result = await logic.IsLargePageAsync("http://www.example.com/");
    Assert.IsFalse(result);
  }

  [TestMethod]
  public async Task UrlCount1025_IsLarge()
  {
    IMyService service = new MyServiceStub {
DownloadAndCountBytesAsyncResult = 1025 };
    var logic = new UseMyService(service);
    var result = await logic.IsLargePageAsync("http://www.example.com/");
    Assert.IsTrue(result);
  }
}
```

Figure 4 **Service with an Asynchronous Factory Method**

```
interface IUniversalAnswerService
{
  int Answer { get; }
}

class UniversalAnswerService : IUniversalAnswerService
{
  private UniversalAnswerService()
  {
  }

  private async Task InitializeAsync()
  {
    await Task.Delay(TimeSpan.FromSeconds(2));
    Answer = 42;
  }

  public static async Task<UniversalAnswerService> CreateAsync()
  {
    var ret = new UniversalAnswerService();
    await ret.InitializeAsync();
    return ret;
  }

  public int Answer { get; private set; }
}
```

Constructors can't be async, but static methods can. One way of faking an asynchronous constructor is to implement an asynchronous factory method, as shown in **Figure 4**.

I really like the asynchronous factory approach because it can't be misused. Calling code can't invoke the constructor directly; it must use the factory method to get an instance, and the instance is fully initialized before it's returned. However, this can't be used in some scenarios. As of this writing, inversion of control (IoC) and dependency injection (DI) frameworks don't understand any conventions for asynchronous factory methods. If you're injecting your services using an IoC/DI container, you'll need an alternative approach.

Asynchronous Resources

In some cases, asynchronous initialization is needed only once, to initialize shared resources. Stephen Toub developed an AsyncLazy<T> type (bit.ly/1cVC3nb), which is also available as a part of my AsyncEx library (bit.ly/1iZBHOW). AsyncLazy<T> combines Lazy<T> with Task<T>. Specifically, it's a Lazy<Task<T>>, a lazy type that supports asynchronous factory methods. The Lazy<T> layer provides thread-safe lazy initialization, ensuring the factory method is only executed once; the Task<T> layer provides asynchronous support, permitting callers to asynchronously wait for the factory method to complete.

Figure 5 presents a slightly simplified definition of AsyncLazy<T>. **Figure 6** shows how AsyncLazy<T> can be used within a type.

This service defines a single shared "resource" that must be constructed asynchronously. Any methods of any instances of this service can depend on that resource and await it directly. The first time the AsyncLazy<T> instance is awaited, it will start the asynchronous factory method once on a thread pool thread. Any other simultaneous access to that same instance from another thread will wait until the asynchronous factory method has been queued to the thread pool.

The synchronous, thread-safe part of the AsyncLazy<T> behavior is handled by the Lazy<T> layer. The time spent blocking is very short: each thread waits only for the factory method to be queued to the thread pool; they don't wait for it to execute. Once the Task<T> is returned from the factory method, then the Lazy<T> layer's job is over. The same Task<T> instance is shared with every await. Neither asynchronous factory methods nor asynchronous lazy initialization will ever expose an instance of T until its asynchronous initialization has completed. This protects against accidental misuse of the type.

AsyncLazy<T> is great for one particular kind of problem: asynchronous initialization of a shared resource. However, it can be awkward to use in other scenarios. In particular, if a service instance needs an asynchronous constructor, you can define an "inner" service type that does the asynchronous initialization and use AsyncLazy<T> to wrap the inner instance within the "outer" service type. But that leads to cumbersome and tedious code, with all methods depending on the same inner instance. In such scenarios, a true "asynchronous constructor" would be more elegant.

A Misstep

Before I go into my preferred solution, I want to point out a somewhat common misstep. When developers are confronted with asynchronous work to do in a constructor (which can't be async), the workaround can be something like the code in **Figure 7**.

But there are some serious problems with this approach. First, there's no way to tell when the initialization has completed; second, any exceptions from the initialization will be handled in the usual async void manner, commonly crashing the application.

Figure 5 **Definition of AsyncLazy<T>**

```
// Provides support for asynchronous lazy initialization.
// This type is fully thread-safe.
public sealed class AsyncLazy<T>
{
  private readonly Lazy<Task<T>> instance;

  public AsyncLazy(Func<Task<T>> factory)
  {
    instance = new Lazy<Task<T>>(() => Task.Run(factory));
  }

  // Asynchronous infrastructure support.
  // Permits instances of this type to be awaited directly.
  public TaskAwaiter<T> GetAwaiter()
  {
    return instance.Value.GetAwaiter();
  }
}
```

Figure 6 **AsyncLazy<T> Used in a Type**

```
class MyServiceSharingAsyncResource
{
  private static readonly AsyncLazy<int> _resource =
    new AsyncLazy<int>(async () =>
    {
       await Task.Delay(TimeSpan.FromSeconds(2));
       return 42;
    });

  public async Task<int> GetAnswerTimes2Async()
  {
    int answer = await _resource;
    return answer * 2;
  }
}
```

Want to win a FREE

Samsung 65” LED Smart TV at Microsoft TechEd 2014?

Just fill out the simple survey on the back of this card and bring to Booth #1837!

Fill out this survey card and take it with you to the LEADTOOLS Booth #1837 for a FREE t-shirt and the chance to win a Samsung 65" LED Smart TV.

Do you read or visit any of the following at least once per month?

- SD Times
- Visual Studio Magazine
- MSDN
- CODE Magazine
- Code Project
- Dream.in.code
- C#Corner
- DaniWeb

What types of development projects are you working on or will work on in the next 12 months? (Circle all that apply)

- Imaging
  - Document imaging
  - PDF
  - OCR/ICR/OMR
  - Barcode
  - Medical Imaging
- Multimedia
- No Imaging Development

What do you use most at work? (Circle all that apply)

Programming Environment:

- VS 2012
- VS 2005 or older
- VS 2010
- Non-Microsoft
- VS 2008

Language:

- C#
- ASP.N+ET
- VB.NET
- JavaScript
- C++
- VB

Have you heard of LEADTOOLS Imaging or Multimedia SDKs before today? If so, where have you seen us?

- I've used it
- My company uses it
- Search engine
- Programming website
- Banner ad
- Someone told me about it
- Magazine ad
- I have not heard of LEADTOOLS before

Do you do any mobile-focused development at work?

- YES
- NO

Email address (please use email associated with TechEd badge): _____

The premier Imaging SDK for all your document, medical and multimedia needs

Document

OCR, Barcode & Forms Recognition

PDF Read, Write & Edit

Cleanup and Preprocessing

Medical

Dicom & PACS

Medical Workstation

HTML5 Zero Footprint Viewing

Multimedia

Playback, Capture & Conversion

MPEG-2 Transport Stream

Distributed Transcoding

C++ C# JavaScript VB Objective-C Java

.NET Windows API WinRT Linux iOS OS X Android HTML5

DOWNLOAD OUR 60 DAY EVALUATION
WWW.LEADTOOLS.COM

LEAD TECHNOLOGIES INCORPORATED

SALES@LEADTOOLS.COM
800.637.1840

Figure 7 **Workaround When Confronted with Async Work To Do in a Constructor**

```
class BadService
{
  public BadService()
  {
    InitializeAsync();
  }

  // BAD CODE!!
  private async void InitializeAsync()
  {
    await Task.Delay(TimeSpan.FromSeconds(2));
    Answer = 42;
  }

  public int Answer { get; private set; }
}
```

If InitializeAsync was async task instead of async void, the situation would barely be improved: There would still be no way to tell when the initialization completed, and any exceptions would be silently ignored. There's a better way!

The Asynchronous Initialization Pattern

Most reflection-based creation code (IoC/DI frameworks, Activator.CreateInstance and so forth) assumes your type has a constructor, and constructors can't be asynchronous. If you're in this situation, you're forced to return an instance that hasn't been (asynchronously) initialized. The purpose of the asynchronous initialization pattern is to provide a standard way of handling this situation, to mitigate the problem of uninitialized instances.

First, I define a "marker" interface. If a type needs asynchronous initialization, it implements this interface:

```
/// <summary>
/// Marks a type as requiring asynchronous initialization and
/// provides the result of that initialization.
/// </summary>
public interface IAsyncInitialization
{
  /// <summary>
  /// The result of the asynchronous initialization of this instance.
  /// </summary>
  Task Initialization { get; }
}
```

At first glance, a property of type Task feels odd. I believe it's appropriate, though, because the asynchronous operation (initializing the instance) is an instance-level operation. So the Initialization property pertains to the instance as a whole.

Figure 8 **Service Implementing the InitializeAsync Method**

```
class UniversalAnswerService : IUniversalAnswerService, IAsyncInitialization
{
  public UniversalAnswerService()
  {
    Initialization = InitializeAsync();
  }

  public Task Initialization { get; private set; }

  private async Task InitializeAsync()
  {
    await Task.Delay(TimeSpan.FromSeconds(2));
    Answer = 42;
  }

  public int Answer { get; private set; }
}
```

When I implement this interface, I prefer to do so with an actual async method, which I name InitializeAsync by convention, as **Figure 8** shows:

The constructor is quite simple; it starts the asynchronous initialization (by calling InitializeAsync) and then sets the Initialization property. That Initialization property provides the results of the InitializeAsync method: when InitializeAsync completes, the Initialization task completes, and if there are any errors, they will be surfaced through the Initialization task.

When the constructor completes, the initialization might not yet be complete, so the consuming code has to be careful. The code using the service has the responsibility to ensure the initialization is complete before calling any other methods. The following code creates and initializes a service instance:

```
async Task<int> AnswerTimes2Async()
{
  var service = new UniversalAnswerService();
  // Danger! The service is uninitialized here; "Answer" is 0!
  await service.Initialization;
  // OK, the service is initialized and Answer is 42.
  return service.Answer * 2;
}
```

In a more realistic IoC/DI scenario, the consuming code only gets an instance implementing IUniversalAnswerService, and has to test whether it implements IAsyncInitialization. This is a useful technique; it allows the asynchronous initialization to be an implementation detail of the type. For example, stub types will probably not use asynchronous initialization (unless you're actually testing that the consuming code will wait for the service to be initialized). The following code is a more realistic use of my answer service:

```
async Task<int> AnswerTimes2Async(IUniversalAnswerService service)
{
  var asyncService = service as IAsyncInitialization;
  if (asyncService != null)
    await asyncService.Initialization;
  return service.Answer * 2;
}
```

Before continuing with the asynchronous initialization pattern, I should point out an important alternative. It's possible to expose the service members as asynchronous methods that internally await the initialization of their own objects. **Figure 9** shows what this kind of object would look like.

Figure 9 **Service That Awaits Its Own Initialization**

```
class UniversalAnswerService
{
  private int _answer;

  public UniversalAnswerService()
  {
    Initialization = InitializeAsync();
  }

  public Task Initialization { get; private set; }

  private async Task InitializeAsync()
  {
    await Task.Delay(TimeSpan.FromSeconds(2));
    _answer = 42;
  }

  public Task<int> GetAnswerAsync()
  {
    await Initialization;
    return _answer;
  }
}
```

Extreme Performance Linear Scalability

NCache™
100% .NET

For .NET & Java Apps
(Windows Azure & Amazon AWS Supported)

JvCache™
100% Java

Cache data, reduce expensive database trips, and scale your apps to extreme transaction processing (XTP) with NCache. JvCache is 100% native Java implementation of NCache.

In-Memory Distributed Cache

- Extremely fast & linearly scalable with 100% uptime
- Mirrored, Replicated, Partitioned, and Client Cache
- NHibernate & Entity Framework Level-2 Cache

ASP.NET Optimization in Web Farms

- ASP.NET Session State storage
- ASP.NET View State cache
- ASP.NET Output Cache provider
- ASP.NET JavaScript merge/minify

Runtime Data Sharing

- Powerful event notifications for pub/sub data sharing

Version 4.3 Released

Memcached Wrapper Provided

www.alachisoft.com
sales@alachisoft.com

Download a FREE trial!

US: +1 (925) 236 3830
UK: +44 (20) 7993 8327

I like this approach because it isn't possible to misuse an object that hasn't yet been initialized. However, it limits the API of the service, because any member that depends on initialization must be exposed as an asynchronous method. In the preceding example, the Answer property was replaced with a GetAnswerAsync method.

Composing the Asynchronous Initialization Pattern

Let's say I'm defining a service that depends on several other services. When I introduce the asynchronous initialization pattern for my services, any of those services might require asynchronous initialization. The code for checking whether those services implement IAsyncInitialization can get somewhat tedious, but I can easily define a helper type:

```
public static class AsyncInitialization
{
  public static Task EnsureInitializedAsync(IEnumerable<object> instances)
  {
    return Task.WhenAll(
      instances.OfType<IAsyncInitialization>().Select(x => x.Initialization));
  }

  public static Task EnsureInitializedAsync(params object[] instances)
  {
    return EnsureInitializedAsync(instances.AsEnumerable());
  }
}
```

The helper methods take any number of instances of any type, filter out any that don't implement IAsyncInitialization, and then asynchronously wait for all the Initialization tasks to complete.

With these helper methods in place, creating a compound service is straightforward. The service in **Figure 10** takes two instances of the answer service as dependencies, and averages their results.

There are a few important takeaways to keep in mind when composing services. First, because asynchronous initialization is an implementation detail, the composed service can't know whether any of its dependencies require asynchronous initialization. If none of the dependencies require asynchronous initialization, then neither would the compound service. But because it can't know, the compound service must declare itself as requiring asynchronous initialization.

Figure 10 **Service That Averages Results of the Answer Service As Dependencies**

```
interface ICompoundService
{
  double AverageAnswer { get; }
}

class CompoundService : ICompoundService, IAsyncInitialization
{
  private readonly IUniversalAnswerService _first;
  private readonly IUniversalAnswerService _second;

  public CompoundService(IUniversalAnswerService first,
    IUniversalAnswerService second)
  {
    _first = first;
    _second = second;
    Initialization = InitializeAsync();
  }

  public Task Initialization { get; private set; }

  private async Task InitializeAsync()
  {
    await AsyncInitialization.EnsureInitializedAsync(_first, _second);
    AverageAnswer = (_first.Answer + _second.Answer) / 2.0;
  }

  public double AverageAnswer { get; private set; }
}
```

Don't worry too much about the performance implications of this; there will be some extra memory allocations for the asynchronous structures, but the thread will not behave asynchronously. Await has a "fast path" optimization that comes into play whenever code awaits a task that's already complete. If the dependencies of a compound service don't require asynchronous initialization, the sequence passed to Task.WhenAll is empty, causing Task.WhenAll to return an already-completed task. When that task is awaited by CompoundService.InitializeAsync, it won't yield execution because the task has already completed. In this scenario, InitializeAsync completes synchronously, before the constructor completes.

A second takeaway is that it's important to initialize all dependencies before the compound InitializeAsync returns. This ensures the compound type's initialization is fully complete. Also, error handling is natural—if a dependent service has an initialization error, those errors propagate up from EnsureInitializedAsync, causing the compound type's InitializeAsync to fail with the same error.

The final takeaway is that the compound service is not a special kind of type. It's just a service that supports asynchronous initialization, just like any other kind of service. Any of these services can be mocked for testing, whether they support asynchronous initialization or not.

Wrapping Up

The patterns in this article can apply to any type of application; I've used them in ASP.NET and console, as well as MVVM applications. My own favorite pattern for asynchronous construction is the asynchronous factory method; it's very simple and can't be abused by consuming code because it never exposes an uninitialized instance. However, I've also found the asynchronous initialization pattern quite useful when working in scenarios where I can't (or don't want to) create my own instances. The AsyncLazy<T> pattern also has its place, when there are shared resources that require asynchronous initialization.

The asynchronous service patterns are more established than the MVVM patterns I introduced earlier in this series. The pattern for asynchronous data binding and the various approaches for asynchronous commands are both fairly new, and they certainly have room for improvement. The asynchronous service patterns, in contrast, have been used more widely. However, the usual caveats apply: These patterns are not gospel; they're just techniques I've found useful and wanted to share. If you can improve on them or tailor them to your application's needs, please, go right ahead! I hope these articles have been helpful to introduce you to asynchronous MVVM patterns, and, even more, that they've encouraged you to extend them and explore your own asynchronous patterns for UIs. ■

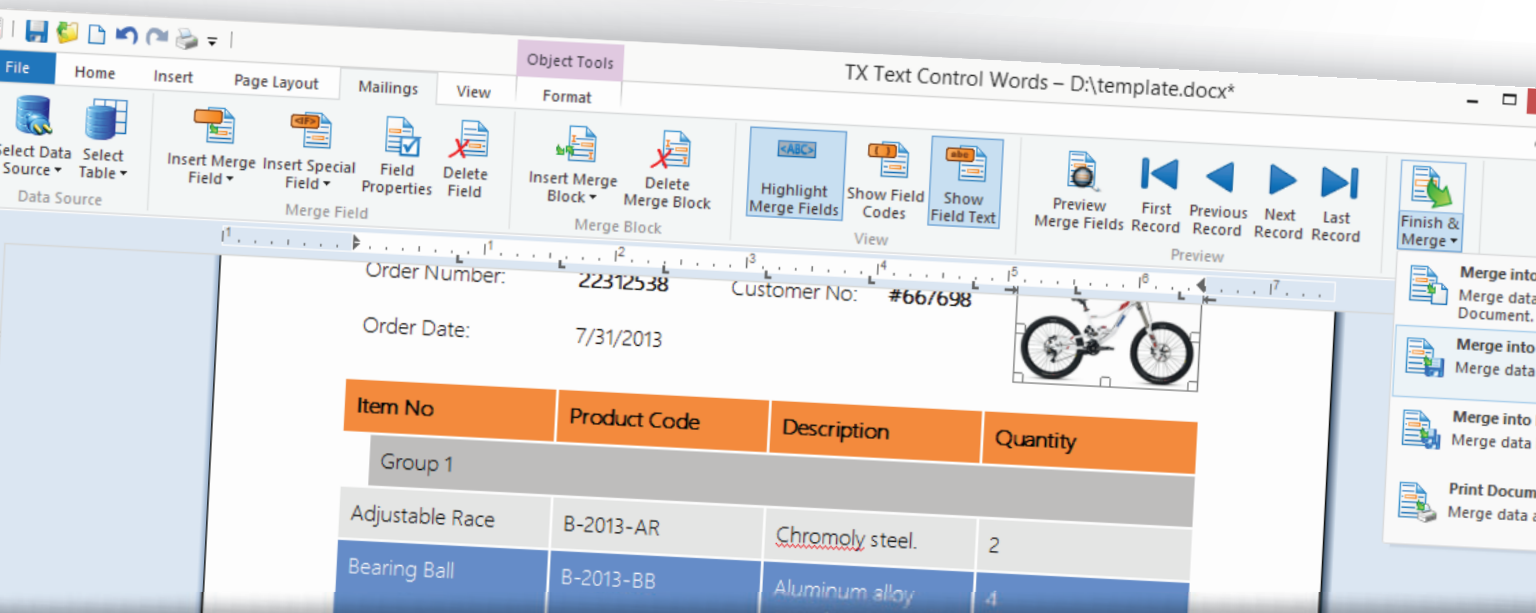
Stephen Cleary *is a husband, father and programmer living in northern Michigan. He has worked with multithreading and asynchronous programming for 16 years and has used async support in the Microsoft .NET Framework since the first CTP. His homepage, including his blog, is at stephencleary.com.*

Thanks to the following Microsoft technical experts for reviewing this article: *James McCaffrey and Stephen Toub*

TX TEXT CONTROL®
word processing components

Creating a report is as easy as writing a letter

- Reuse MS Word documents as your reporting templates
- Create encrypted and print-ready Adobe PDF and PDF/A
- Royalty-free WYSIWYG template designer
- Powerful and dynamic 2D/3D charting support
- Easy database connections and master-detail nested blocks
- 1D/2D barcode support including QRCode, IntelligentMail, EAN



www.textcontrol.com/reporting

txtextcontrol

US: +1 855-533-TEXT
EU: +49 421 427 067-10

Visual Studio Microsoft Partner

Reporting | Rich Text Editing | Spell Checking | Barcodes | PDF Reflow

Software Development with Feature Toggles

Bill Heys

Feature toggles as a software development concept let you pursue parallel, concurrent feature development as an alternative to branching for parallel development (also called feature branches). Feature toggles are sometimes called feature flags, feature switches, feature flippers or conditional features. Feature toggles let you continuously integrate features while they're under development. You can use feature toggles to hide, disable or enable individual features during runtime.

As with all software development techniques, you would use feature toggles in conjunction with version control (such as Microsoft Team Foundation Server). Using feature toggles by themselves doesn't imply eliminating all branches from a comprehensive version control plan. One distinction of feature toggles is that all changes are checked into the main branch (mainline) instead of a development branch.

A feature is disabled or hidden from all users until you start feature development. During development, you can enable the feature for development and unit testing, and disable it for all other users. Quality Assurance (QA) testers can also enable features they want to test. Until the feature is complete and fully tested according to the Definition of Done (DoD) and passes quality tests, it will be hidden or disabled in the released software. You can dynamically change the value of a feature toggle without creating and deploying a new build.

Starting with a comparison of feature branches and feature toggles, I'll suggest alternative techniques for implementing feature toggles in code. Using a sample Windows calculator application, I'll illustrate how to hide or show features at run time using feature toggles.

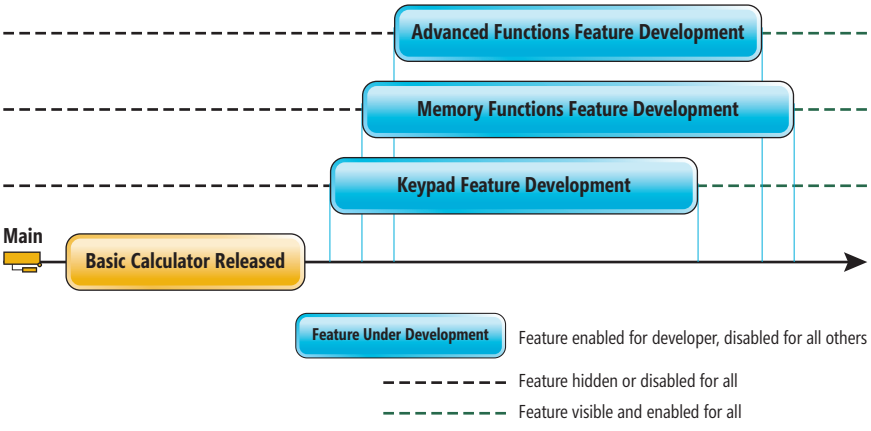


Figure 1 **Feature Toggles Support Continuous Integration of Parallel Development**

BUILD A RELEASE PIPELINE WITH TEAM FOUNDATION SERVER 2012

Produce better software, faster, by employing continuous delivery techniques like versioning, continuous integration and automation in Team Foundation Server 2012 or Team Foundation Server 2013.

bit.ly/1fwNJrh

This article discusses:

- How to use feature toggles to continuously integrate feature code
- The difference between feature branching and feature toggles
- Dynamic software development and deployment

Technologies discussed:

Visual Studio, Team Foundation Server

Continuous Integration

Using feature toggles, you can do continuous integration—all development work is checked into the main branch and continuously integrated with all other code in that branch. On every check-in, code in the main branch is built and tested on a build server using automated Build Verification Tests (BVTs).

As stated in the Microsoft patterns & practices book on continuous delivery ("Building a Release Pipeline with Team Foundation Server 2012" [2013]), "By continuous delivery, we mean that through techniques such as versioning, continuous integration, automation,

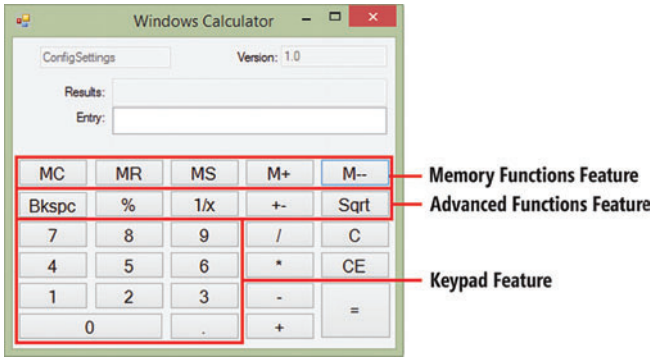


Figure 2 **Sample Windows Calculator Showing Three New Features**

and environment management, you will be able to decrease the time between when you first have an idea and when that idea is realized as software that's in production" (bit.ly/1kFVOKx).

You can use automated unit tests to test your code prior to checking it into the main branch. If a check-in breaks a build on the build server, you'll have to fix the code before the check-in is allowed. If a feature in the build doesn't pass QA testing, you can hide it from deployment or release until it does.

Feature toggles give you runtime isolation of a feature being developed until it's complete, tested and ready for release. Using feature toggles with continuous integration, you work directly in the main branch. Code is checked in to the main branch, while keeping this branch stable for builds and deployments (see **Figure 1**).

Feature Toggles in Use

Figure 2 shows an example of feature toggles at work in a sample Windows Calculator Form. This also illustrates the challenges of parallel concurrent feature development.

This example compares managing parallel development using feature branches to using feature toggles. There are three new features (Keypad, Advanced Functions and Memory Functions) being developed in parallel as the basic calculator is deployed.

Feature Toggle Patterns

There are many usage patterns in which you can use feature toggles to allow for continuous integration of all code, including pending or incomplete code. Using feature toggles can increase a team's development speed by reducing or eliminating the need for parallel development branches and the ensuing branching and merging tasks, which can be extremely time-consuming and error-prone.

Here's a list of typical scenarios where you could consider feature toggles:

- Hiding or disabling new features in the UI
- Hiding or disabling new components in the application
- Versioning an interface
- Extending an interface
- Supporting multiple versions of a component
- Adding a new feature to an existing application
- Enhancing an existing feature in an existing application

Branching for Feature Isolation

With a feature branching strategy, your team can isolate concurrent or parallel development of specific features—or groups of features—into separate feature branches. Feature branches give you significant flexibility in terms of when to move a release into production. You don't have to wait until all features are ready in order to do a full deployment.

You can easily merge individual features or groups of features into the main branch when they're complete according to the DoD. You can create new feature branches as needed, and delete them once the feature is complete. Features aren't merged to the main branch until they meet the DoD. Following the example, **Figure 3** shows three new feature branches providing isolation during parallel development.

For more guidance on various feature branching scenarios, you can refer to the ALM Rangers e-book on branching strategies at aka.ms/vsarsolutions.

Feature Toggles vs. Feature Branches

Using feature toggles lets you and your team pursue continuous integration, continuous deployment and continuous release practices. These practices permit more frequent feature code integration as they're developed in parallel when compared to feature branches. At the same time, feature toggles also support runtime isolation of features under development and not ready for release.

With feature toggles, all pending changes are checked into the main branch. Each check-in is pended until an automated build process builds all code from the main branch on a build server, and successfully runs automated BVTs. This is the process known as continuous integration.

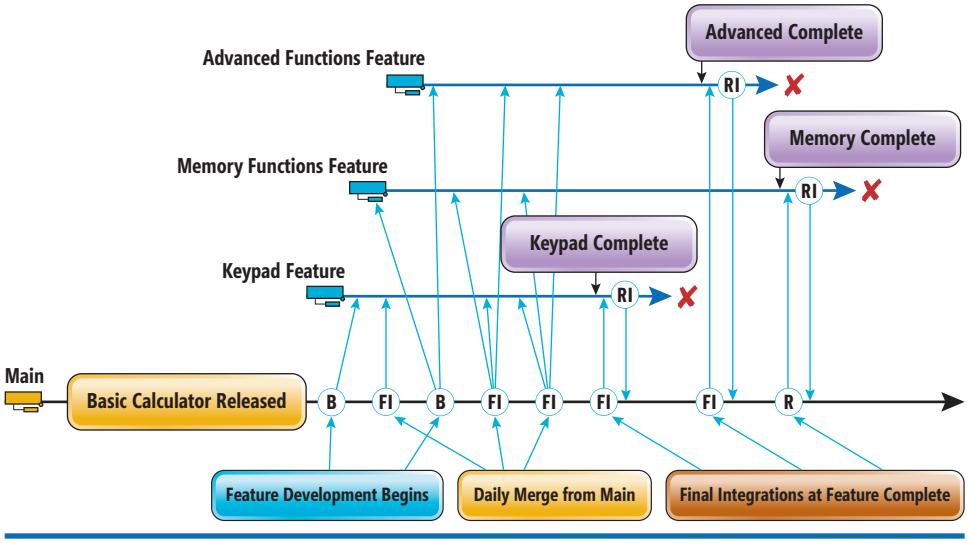
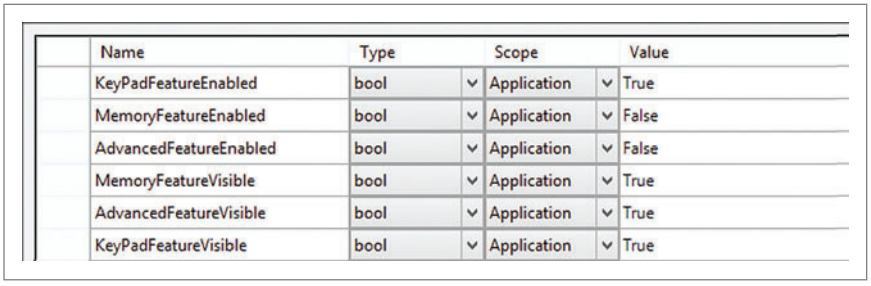


Figure 3 **Branching for Feature Isolation**

At run time, feature toggles hide or bypass features that are in the build, but not ready for release. Depending on how you plan to implement a feature, the effort to use feature toggles may sometimes be overly complex. On the other hand, hiding features from the UI can be straightforward.

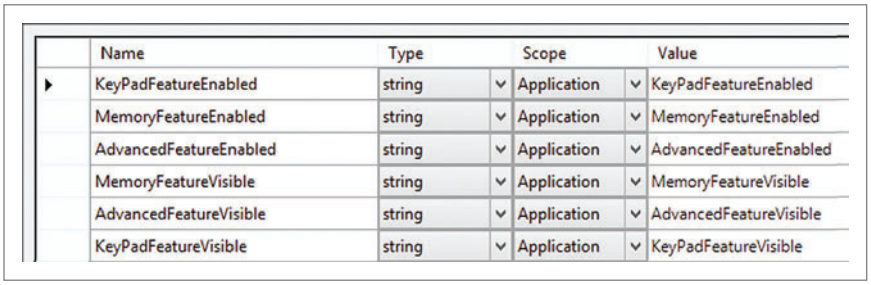
When using feature branches, all development is checked into the associated feature branch and isolated from code in the main branch or other feature branches. You can't merge a new or enhanced feature with the main branch or other feature branches until the feature is Code Complete, passes required quality tests or meets the DoD. Only at that point is the feature integrated (merged) with the main branch. Therefore, code isolation and continuous integration are at somewhat opposite ends of the "integration spectrum."

Enabling a new feature in a deployment is relatively easy and risk free. Simply activate the associated feature toggles to visible and enabled. The effort to release a new feature using feature branches is much more complicated. You have to merge the feature into the main branch. This is often a time-consuming and challenging process.



Name	Type	Scope	Value
KeyPadFeatureEnabled	bool	Application	True
MemoryFeatureEnabled	bool	Application	False
AdvancedFeatureEnabled	bool	Application	False
MemoryFeatureVisible	bool	Application	True
AdvancedFeatureVisible	bool	Application	True
KeyPadFeatureVisible	bool	Application	True

Figure 4 **This Sample ConfigSettings File Stores Feature Toggle State for the Windows Calculator**



Name	Type	Scope	Value
KeyPadFeatureEnabled	string	Application	KeyPadFeatureEnabled
MemoryFeatureEnabled	string	Application	MemoryFeatureEnabled
AdvancedFeatureEnabled	string	Application	AdvancedFeatureEnabled
MemoryFeatureVisible	string	Application	MemoryFeatureVisible
AdvancedFeatureVisible	string	Application	AdvancedFeatureVisible
KeyPadFeatureVisible	string	Application	KeyPadFeatureVisible

Figure 5 **Feature Toggle Names Stored in the Database Table As Rows**

Removing a Feature

In some projects, you'll have to remove or back out incomplete features. Rolling back individual changesets to roll back a feature (cherry picking changes) is also complicated, labor-intensive and might require significant code rework.

It's imperative to isolate features from the main branch until you decide to release those features. In any event, backing out a feature from a release just before the release is scheduled is risky and error-prone. Using feature toggles, you check all code for all features into the main branch. Feature toggles simply hide or disable select features from runtime or release.

Implementation Choices

There are multiple approaches to persisting the state of an application's feature toggles. Here are two fairly straightforward approaches:

1. Use Application Settings to define feature toggles (deployed in the XML Application Configuration file when the application is built).
2. Store feature toggles as rows in a database table.

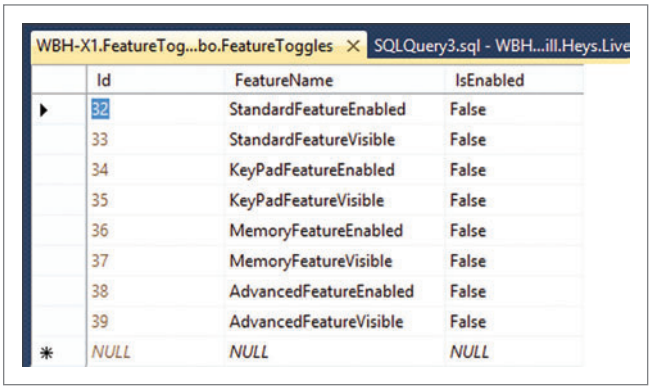
Define Feature Toggles Not only is it easier to define feature toggles in the configuration file using the Settings Designer, but you don't need to do anything in your application to save the state of the feature toggles when the application ends. The sample Settings file shown in **Figure 4** defines feature toggles.

Store Feature Toggles As Database Rows To use a feature toggle database table to store the feature toggle state, the application will require two settings files. The CommonSettings settings file defines options used by the application. The DatabaseSettings settings file controls whether feature toggles are defined and persisted in a database table (when True) or in a Settings file (when False). The FeatureToggleSettings file provides the names for each of the feature toggles you'll store in the database (see **Figure 5**).

The approach used in the sample Windows Calculator is to dynamically create feature toggle rows. When the application runs for the first time, the table will be empty. At run time, all feature toggle names are checked to see if they're enabled. If any aren't already in the table, the application will add them to the table. The sample FeatureToggle table in **Figure 6** shows all feature toggle rows added.

Implementation-Independent Concepts

When you use a feature toggle to disable a feature, code in the application hides or disables the controls based on the value of the feature toggle (True/False) associated with that feature. You can choose to hide the feature completely from the UI for features under development, or to show the feature as disabled. When you release the feature, it will be visible and enabled.



WBH-X1.FeatureTog...bo.FeatureToggles | SQLQuery3.sql - WBH...ill.Heys.Live

Id	FeatureName	IsEnabled
32	StandardFeatureEnabled	False
33	StandardFeatureVisible	False
34	KeyPadFeatureEnabled	False
35	KeyPadFeatureVisible	False
36	MemoryFeatureEnabled	False
37	MemoryFeatureVisible	False
38	AdvancedFeatureEnabled	False
39	AdvancedFeatureVisible	False
NULL	NULL	NULL

Figure 6 **FeatureToggle Table After Initial Run**

Figure 7 shows the Windows Calculator with only the keypad feature visible and enabled. The memory functions and advanced functions features are visible, but disabled.

The first step in this sample application creates a new FeatureToggleDemo database, followed by creating a simple FeatureToggle table. When you download the sample application, you will find three SQL files for: Creating the FeatureToggle Database, creating the FeatureToggle Table, and creating the two stored procedures used by the application to load values from and save them back to the FeatureToggle table.

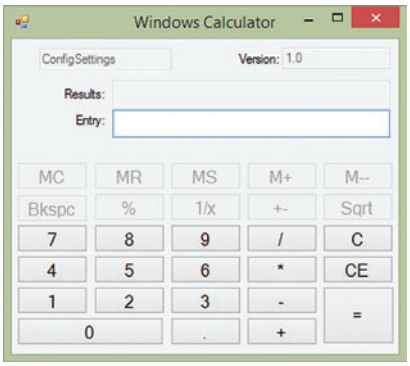


Figure 7 **Windows Calculator with Keypad Feature Visible and Enabled, and the Memory and Advanced Functions Features Visible and Disabled**

After creating the FeatureToggle table, there's no need to add any FeatureToggle rows prior to running the application for the first time. The application uses a FeatureToggle class library to encapsulate all of the application logic for accessing and updating the feature toggles.

The following code shows private fields used to hold the state of the feature toggles:

```
private Boolean _IsKeyPadFeatureEnabled = false;
private Boolean _IsKeyPadFeatureVisible = false;
private Boolean _IsMemoryFeatureEnabled = false;
private Boolean _IsMemoryFeatureVisible = false;
private Boolean _IsAdvancedFeatureVisible = false;
private Boolean _IsAdvancedFeatureEnabled = false;
```

The code in **Figure 8** shows how to set Private Fields from feature toggle values stored in a ConfigSettings file.

The code in **Figure 9** shows how to set Private Fields from feature toggle values stored in the FeatureToggle database table.

Figure 8 **Feature Toggle Values Stored in a ConfigSettings File**

```
private void GetFeatureTogglesFromSettings()
{
  _IsKeyPadFeatureEnabled = Properties.ConfigSettings.Default.KeyPadFeatureEnabled;
  _IsKeyPadFeatureVisible = Properties.ConfigSettings.Default.KeyPadFeatureVisible;
  _IsMemoryFeatureEnabled = Properties.ConfigSettings.Default.MemoryFeatureEnabled;
  _IsMemoryFeatureVisible = Properties.ConfigSettings.Default.MemoryFeatureVisible;
  _IsAdvancedFeatureEnabled = Properties.ConfigSettings.Default.
AdvancedFeatureEnabled;
  _IsAdvancedFeatureVisible = Properties.ConfigSettings.Default.
AdvancedFeatureVisible;
  tbMode.Text = Constants.configSettings;
}
```

Figure 9 **Set Private Fields from Stored Feature Toggle Values Stored in a Database Table**

```
private void GetFeatureTogglesFromStore()
{
  _IsKeyPadFeatureEnabled = FeatureToggles.FeatureToggles.IsEnabled(
    Properties.FeatureToggleSettings.Default.KeyPadFeatureEnabled);
  _IsKeyPadFeatureVisible = FeatureToggles.FeatureToggles.IsEnabled(
    Properties.FeatureToggleSettings.Default.KeyPadFeatureVisible);
  _IsMemoryFeatureEnabled = FeatureToggles.FeatureToggles.IsEnabled(
    Properties.FeatureToggleSettings.Default.MemoryFeatureEnabled);
  _IsMemoryFeatureVisible = FeatureToggles.FeatureToggles.IsEnabled(

Properties.FeatureToggleSettings.Default.MemoryFeatureVisible);
  _IsAdvancedFeatureEnabled = FeatureToggles.FeatureToggles.IsEnabled(
    Properties.FeatureToggleSettings.Default.AdvancedFeatureEnabled);
  _IsAdvancedFeatureVisible = FeatureToggles.FeatureToggles.IsEnabled(
    Properties.FeatureToggleSettings.Default.AdvancedFeatureVisible);
  tbMode.Text = Constants.databaseSettings;
}
```

The code in **Figure 10** shows how the application evaluates feature toggles and sets the Visible and Enabled properties of the Windows Calculator feature using wrapper methods.

The following code shows a wrapper function to hide or show UI elements associated with the Advanced Functions feature:

```
private void SetAdvancedFeatureVisible(bool state)
{
  this.btnBkspc.Visible = state;
  this.btnSqrt.Visible = state;
  this.btnPercent.Visible = state;
  this.btnReciprocal.Visible = state;
  this.btnPlusMinus.Visible = state;
}
```

The Windows Calculator takes advantage of a reusable class library to encapsulate all processing associated with the FeatureToggle table. Using this class library, the code that uses the ConfigSettings file is almost identical to the code using a FeatureToggle database table.

Wrapping Up

The benefit of feature toggles is that new or enhanced features are checked in to the main branch, so they can be continuously integrated and tested with the existing codebase at build time. Previously, code from new or enhanced features was generally not integrated with the codebase until closer to the release deadline, which is risky, challenging and fraught with peril. Both feature toggles and feature branches are viable approaches you can use to achieve a stable release with only completed features that have passed necessary QA tests. The choice to use one or the other depends on your needs and your development processes in place. ■

Bill Heys *is a senior ALM consultant with Design Process and Solutions. He was previously a senior ALM consultant with Microsoft. He is a member of the Microsoft Visual Studio ALM Rangers, lead developer on the Rangers Branch and Merge guidance and a contributor to the release of the Rangers Version Control guidance. Reach him at bill.heys@live.com.*

THANKS to the following technical experts for reviewing this article: *Michael Fourie (independent consultant), Micheal Learned (Microsoft), Matthew Mitrik (Microsoft) and Willy-Peter Schaub (Microsoft)*

Figure 10 **Using Wrapper Methods to Set the Visible and Enabled Properties of the Windows Calculator**

```
private void EvaluateFeatureToggles()
  {
    this.tbVersion.Text = Properties.CommonSettings.Default.Version;
    if (Properties.CommonSettings.Default.DatabaseSettings)
    {
      GetFeatureTogglesFromStore();
    }
    else
    {
      GetFeatureTogglesFeatureToggleFromSettings();
    }
    if (_IsAdvancedFeatureEnabled)
    {
      _IsAdvancedFeatureVisible = true;
    }
    SetAdvancedFeatureEnabled(_IsAdvancedFeatureEnabled);
    SetAdvancedFeatureVisible(_IsAdvancedFeatureVisible);
  }
```

Spreadsheets Made Easy.

Fastest Calculations

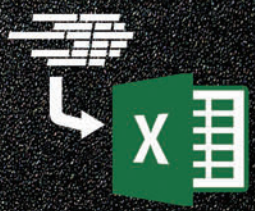
Evaluate complex Excel-based models and business rules with the fastest and most complete Excel-compatible calculation engine available.

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

Dangers of Violating SOLID Principles in C#

Brannon B. King

As the process of writing software has evolved from the theoretical realm into a true engineering discipline, a number of principles have emerged. And when I say *principle*, I'm referring to a feature of the computer code that helps maintain the value of that code. *Pattern* refers to a common code scenario, whether good or bad.

For example, you might value computer code that works safely in a multi-threaded environment. You may value computer code that doesn't crash when you modify code in another location. Indeed, you might value many helpful qualities in your computer code, but encounter the opposite on a daily basis.

There have been some fantastic software development principles captured under the SOLID acronym—Single responsibility, Open for extension and closed for modification, Liskov substitution, Interface segregation, and Dependency injection. You should have some familiarity with these principles, as I'll demonstrate a variety of C#-specific patterns that violate these principles. If you're unfamiliar with the SOLID principles, you might want to review them quickly before proceeding. I'll also assume some familiarity with the architectural terms Model and ViewModel.

The SOLID acronym and the principles encompassed within did not originate with me. Thank you, Robert C. Martin, Michael Feathers, Bertrand Meyer, James Coplien and others, for sharing your wisdom with the rest of us. Many other books and blog posts have explored and refined these principles. I hope to help amplify the application of these principles.

Having worked with and trained many junior software engineers, I've discovered there's a large gap between the first professional coding endeavors and sustainable code. In this article, I'll try to bridge that gap in a lighthearted way. The examples are a bit silly with the goal of helping you recognize you can apply the SOLID principles to all forms of software.

The professional development environment brings many challenges for aspiring software engineers. Your schooling has taught you to think about problems from a top-down perspective. You'll take a top-down approach to your initial assignments in the world of hearty, corporate-sized software. You'll soon find your top-level function has grown to an unwieldy size. To make the smallest change requires full working knowledge of the entire system, and there's little to keep it in check. Guiding software principles (of which only a partial set is mentioned here) will help keep the structure from outgrowing its foundation.

This article discusses:

- The SOLID principles of software development
- How to apply those principles to all types of software
- Avoiding inappropriate dependencies in your code

Technologies discussed:

C#, Windows Presentation Foundation

The Single Responsibility Principle

The Single Responsibility Principle is often defined as: An object should only have one reason to change; the longer the file or class, the more difficult it will be to achieve this. With that definition in mind, look at this code:

```
public IList<IList<Nerd>> ComputeNerdClusters(
  List<Nerd> nerds,
  IPlotter plotter = null) {
  ...
  foreach (var nerd in nerds) {
    ...
    if (plotter != null)
      plotter.Draw(nerd.Location, Brushes.PeachPuff, radius: 10);
    ...
  }
  ...
}
```

What's wrong with this code? Is software being written or debugged? It may be that this particular drawing code is only for debugging purposes. It's nice that it's in a service known only by interface, but it doesn't belong. The brush is a good clue. As beautiful and widespread as puffs of peach may be, it's platform-specific. It's outside the type hierarchy of this computational model. There are many ways to segregate the computation and associated debugging utilities. At the very least, you can expose the necessary data through inheritance or events. Keep the tests and test views separate.

Here's another faulty example:

```
class Nerd {
  public int IQ { get; protected set; }
  public double SuspenderTension { get; set; }
  public double Radius { get; protected set; }

  /// <summary>Get books for growing IQ</summary>
  public event Func<Nerd, IBook> InTheMoodForBook;

  /// <summary>Get recommendations for growing Radius</summary>
  public event Func<Nerd, ISweet> InTheMoodForTwink;

  public IList<Nerd> FitNerdsIntoPaddedRoom(
    IList<Nerd> nerds, IList<Point> boundary)
  {
    ...
  }
}
```

What's wrong with this code? It mixes what's called "school subjects." Remember how you learned about different topics in different classes in school? It's important to maintain that separation in the code—not because they're entirely unrelated, but as an organizational effort. In general, don't put any two of these items in the same class: mathematics, models, grammar, views, physical or platform adapters, customer-specific code, and so on.

You can see a general analogy to things you build in school with sculpture, wood and metal. They need measurements, analysis, instruction and so on. The previous example mixes math and model—FitNerdsIntoPaddedRoom doesn't belong. That method could easily be moved to a utility class, even a static one. You shouldn't have to instantiate models in your math test routines.

Figure 1 **Example of Merging Windows Presentation Foundation Renderers into a Single Source**

```
public abstract class RenderDefinition : ViewModelBase
{
  public abstract DataTemplate Template { get; }
  public abstract Style TemplateStyle { get; }
  public abstract bool SourceContains(object o); // For selectors
  public abstract IEnumerable Source { get; }
}

public void LoadItemsControlFromRenderers(
    ItemsControl control,
    IEnumerable<RenderDefinition> defs) {
  control.ItemTemplateSelector = new DefTemplateSelector(defs);
  control.ItemContainerStyleSelector = new DefStyleSelector(defs);
  var compositeCollection = new CompositeCollection();

  foreach (var renderDefinition in defs)
  {
    var container = new CollectionContainer
    {
      Collection = renderDefinition.Source
    };
    compositeCollection.Add(container);
  }
  control.ItemsSource = compositeCollection;
}
```

Here's another multiple responsibilities example:

```
class AvatarBotPath
{
  public IReadOnlyList<ISegment> Segments { get; private set; }
  public double TargetVelocity { get; set; }
  public bool IsReverse { get { return TargetVelocity < 0; } }
  ...
}

public interface ISegment // Elsewhere
{
  Point Start { get; }
  Point End { get; }
  ...
}
```

What's wrong here? Clearly there are two different abstractions represented by a single object. One of them relates to traversing a shape, the other represents the geometric shape itself. This is common in code. You have a representation and separate use-specific parameters that go with that representation.

Inheritance is your friend here. You can move the TargetVelocity and IsReverse properties to an inheritor and capture them in a concise IHasTravelInfo interface. Alternatively, you could add a general collection of features to the shape. Those needing velocity would then query the features collection to see if it's defined on a particular shape. You could also use some other collection mechanism to pair representations with travel parameters.

The Open Closed Principle

That brings us to the next principle: open for extension, closed for modification. How is it done? Preferably not like this:

```
void DrawNerd(Nerd nerd) {
  if (nerd.IsSelected)
    DrawEllipseAroundNerd(nerd.Position, nerd.Radius);
  if (nerd.Image != null)
    DrawImageOfNerd(nerd.Image, nerd.Position, nerd.Heading);
  if (nerd is IHasBelt) // a rare occurrence
    DrawBelt(((IHasBelt)nerd).Belt);
  // Etc.
}
```

What's wrong here? Well, you'll have to modify this method every time a customer needs new things displayed—and they always need new things displayed. Nearly every new software feature requires some sort of UI element. After all, it was the lack of something in the existing interface that prompted the new feature request. The pattern displayed in this method is a good clue, but you can move those if statements into the methods they guard and it won't make the problem go away.

You need a better plan, but how? What will it look like? Well, you have some code that knows how to draw certain things. That's fine. You just need a general procedure for matching those things with the code to draw them. It will essentially come down to a pattern like this:

```
readonly IList<IRenderer> _renderers = new List<IRenderer>();
void Draw(Nerd nerd)
{
  foreach (var renderer in _renderers)
    renderer.DrawIfPossible(_context, nerd);
}
```

There are other ways to add to the list of renderers. The point of the code, however, is to write drawing classes (or classes about drawing classes) that implement a well-known interface. The renderer must have the smarts to determine if it can or should draw anything based on its input. For example, the belt-drawing code can move to its own "belt renderer" that checks for the interface and proceeds if necessary.

You might need to separate the CanDraw from the Draw method, but that won't violate the Open Closed Principle, or OCP. The code using the renderers shouldn't have to change if you add a new renderer. It's that simple. You should also be able to add the new renderer in the correct order. While I'm using rendering as an example, this also applies to handling input, processing data and storing data. This principle has many applications through all types of software. The pattern is more difficult to emulate in Windows Presentation Foundation (WPF), but it's possible. See **Figure 1** for one possible option.

Here's another foul example:

```
class Nerd
{
  public void WriteName(string name)
  {
    var pocketProtector = new PocketProtector();
    WriteNameOnPaper(pocketProtector.Pen, name);
  }

  private void WriteNameOnPaper(Pen pen, string text)
  {
    ...
  }
}
```

What's wrong here? The problems with this code are vast and sundry. The main issue I want to point out is there's no way to override creating the PocketProtector instance. Code like this makes it difficult to write inheritors. You have a few options for dealing with this scenario. You can change the code to:

- Make the WriteName method virtual. That would also require you make WriteNameOnPaper protected to meet the goal of instantiating a modified pocket protector.
- Make the WriteNameOnPaper method public, but that will maintain the broken WriteName method on your inheritors. This isn't a good option unless you get rid of WriteName, in which case the option devolves into passing an instance of PocketProtector into the method.
- Add an additional protected virtual method whose sole purpose is to construct the PocketProtector.
- Give the class a generic type T that's a type of PocketProtector and construct it with some kind of object factory. Then you'll have the same need to inject the object factory.
- Pass an instance of PocketProtector to this class in its constructor or via a public property, instead of constructing it within the class.

The last option listed is generally the best plan, assuming you can reuse PocketProtector. The virtual creation method is also a good and easy option.

You should consider which methods to make virtual to accommodate the OCP. That decision is often left until the last minute: "I'll make the methods virtual when I need to call them from an inheritor I don't have at the moment." Others may choose to make every method virtual, hoping that will allow extenders the ability to work around any oversight in the initial code.

Both approaches are wrong. They exemplify an inability to commit to an open interface. Having too many virtual methods limits your ability to change the code later. A lack of methods you can override limits the extensibility and reusability of the code. That limits its usefulness and lifespan.

Figure 2 **Ambiguous Method Footprint**

```
interface INerd {
  public int Smartness { get; set; }
}

static class Program
{
  public static string RecallSomeDigitsOfPi<T>(
    this IList<T> nerdSmartnesses) where T : int
  {
    var smartest = nerdSmartnesses.Max();
    return Math.PI.ToString("F" + Math.Min(14, smartest));
  }

  public static string RecallSomeDigitsOfPi<T>(
    this IList<T> nerds) where T : INerd
  {
    var smartest = nerds.OrderByDescending(n => n.Smartness).First();
    return Math.PI.ToString("F" + Math.Min(14, smartest.Smartness));
  }

  static void Main(string[] args)
  {
    IList<int> list = new List<int> { 2, 3, 4 };
    var digits = list.RecallSomeDigitsOfPi();
    Console.WriteLine("Digits: " + digits);
  }
}
```

Here's another common example of OCP violations:

```
class Nerd
{
  public void DanceTheDisco()
  {
    if (this is ChildOfNerd)
        throw new CoordinationException("Can't");
    ...
  }
}

class ChildOfNerd : Nerd { ... }
```

What's wrong here? The Nerd has a hard reference to its child type. That's painful to see, and an unfortunately common mistake for junior developers. You can see it violates the OCP. You have to modify multiple classes to enhance or refactor ChildOfNerd.

> Base classes should never directly reference their inheritors.

Base classes should never directly reference their inheritors. Inheritor functionality is then no longer consistent among inheritors. A great way to avoid this conflict is to put inheritors of a class in separate projects. That way the structure of the project reference tree will disallow this unfortunate scenario.

This issue isn't limited to parent-child relationships. It exists with peer classes as well. Suppose you have something like this:

```
class NerdsInAnArc
{
  public bool Intersects(NerdsInAnLine line)
  {
    ...
  }
  ...
}
```

Arcs and lines are typically peers in the object hierarchy. They shouldn't know any non-inherited intimate details about each other, as those details are often needed for optimal intersection

.NET TOOLS FOR DEV PROS

Whether you're building the most modern touch-enabled apps or maintaining and updating legacy applications, our flagship product, Studio Enterprise, helps to deliver rich, responsive, desktop and web apps on time and under budget.

New 2014 v1 Release!

DOWNLOAD YOUR FREE TRIAL

componentone.com/se

ComponentOne®
a division of GrapeCity®

algorithms. Keep yourself free to modify one without having to change the other. This again brings up a single-responsibility violation. Are you storing arcs or analyzing them? Put analysis operations in their own utility class.

If you need this particular cross-peer ability, then you'll need to introduce an appropriate interface. Follow this rule to avoid the cross-entity confusion: You should use the "is" keyword with an abstraction instead of a concrete class. You could potentially craft an IIntersectable or INerdsInAPattern interface for the example, although you'd likely still defer to some other intersection utility class for analyzing data exposed on that interface.

The Liskov Substitution Principle

The Liskov Substitution Principle defines some guidelines for maintaining inheritor substitution. Passing an object's inheritor in place of the base class shouldn't break any existing functionality in the called method. You should be able to substitute all implementations of a given interface with each other.

C# doesn't allow modifying return types or parameter types in overriding methods (even if the return type is an inheritor of the return type in the base class). Therefore, it won't struggle with the most common substitution violations: contravariance of method arguments (overriders must have the same or base types of parent methods) and covariance of return types (return types in overriding methods must be the same or an inheritor of the return types in the base class). However, it's common to try to work around this limitation:

```
class Nerd : Mammal {
  public double Diopter { get; protected set; }

  public Nerd(int vertebrae, double diopter)
    : base(vertebrae) { Diopter = diopter; }

  protected Nerd(Nerd toBeCloned)
    : base (toBeCloned) { Diopter = toBeCloned.Diopter; }

  // Would prefer to return Nerd instead:
  // public override Mammal Clone() { return new Nerd(this); }

  public new Nerd Clone() { return new Nerd(this); }
}
```

What's wrong here? The behavior of the object changes when called with an abstraction reference. The clone method new isn't virtual, and therefore isn't executed when using a Mammal reference. The keyword new in the method declaration context is supposedly a feature. If you don't control the base class, though, how can you guarantee proper execution?

C# has a few workable alternatives, although they're still somewhat distasteful. You can use a generic interface (something like IComparable<T>) to implement explicitly in every inheritor. However, you'll still need a virtual method that does the actual cloning operation. You need this so your clone matches the derived type. C# also supports the Liskov standard on contravariance of return types and covariance of method arguments when using events, but that won't help you change the exposed interface through class inheritance.

Judging from that code, you might think C# includes the return type in the method footprint the class method resolver is using. That's incorrect—you can't have multiple overrides with different return types, but the same name and input types. Method constraints are also ignored for method resolution. **Figure 2** shows an example of syntactically correct code that won't compile due to method ambiguity.

The code in **Figure 3** shows how the ability to substitute might be broken. Consider your inheritors. One of them could modify the isMoonWalking field at random. If that were to happen, the base class runs the risk of missing a critical cleanup section. The isMoonWalking field should be private. If inheritors need to know, there should be a protected getter property that provides access, but not modification.

Wise and occasionally pedantic programmers will take this a step further. Seal the mouse handlers (or any other method that relies on or modifies private state) and let inheritors use events or other virtual methods that aren't must-call methods. The pattern of requiring a base call is admissible, but not ideal. We've all forgotten to call expected base methods on occasion. Don't allow inheritors to break the encapsulated state.

Liskov Substitution also requires inheritors to not throw new exception types (although inheritors of exceptions already thrown in the base class are fine). C# has no way to enforce this.

The Interface Segregation Principle

Each interface should have a specific purpose. You shouldn't be forced to implement an interface when your object doesn't share that purpose. By extrapolation, the larger the interface, the more likely it includes methods that not all implementers can achieve. That's the essence of the Interface Segregation Principle. Consider an old and common interface pair from the Microsoft .NET Framework:

```
public interface ICollection<T> : IEnumerable<T> {
  void Add(T item);
  void Clear();
  bool Contains(T item);
  void CopyTo(T[] array, int arrayIndex);
  bool Remove(T item);
}

public interface IList<T> : ICollection<T> {
  T this[int index] { get; set; }
  int IndexOf(T item);
  void Insert(int index, T item);
  void RemoveAt(int index);
}
```

The interfaces are still somewhat useful, but there's an implicit assumption that if you're using these interfaces, you want to modify the collections. Oftentimes, whoever creates these data collections wants to prevent anyone from modifying the data. It's actually very useful to separate interfaces into sources and consumers.

Many data stores would like to share a common, indexable non-writable interface. Consider data analysis or data searching

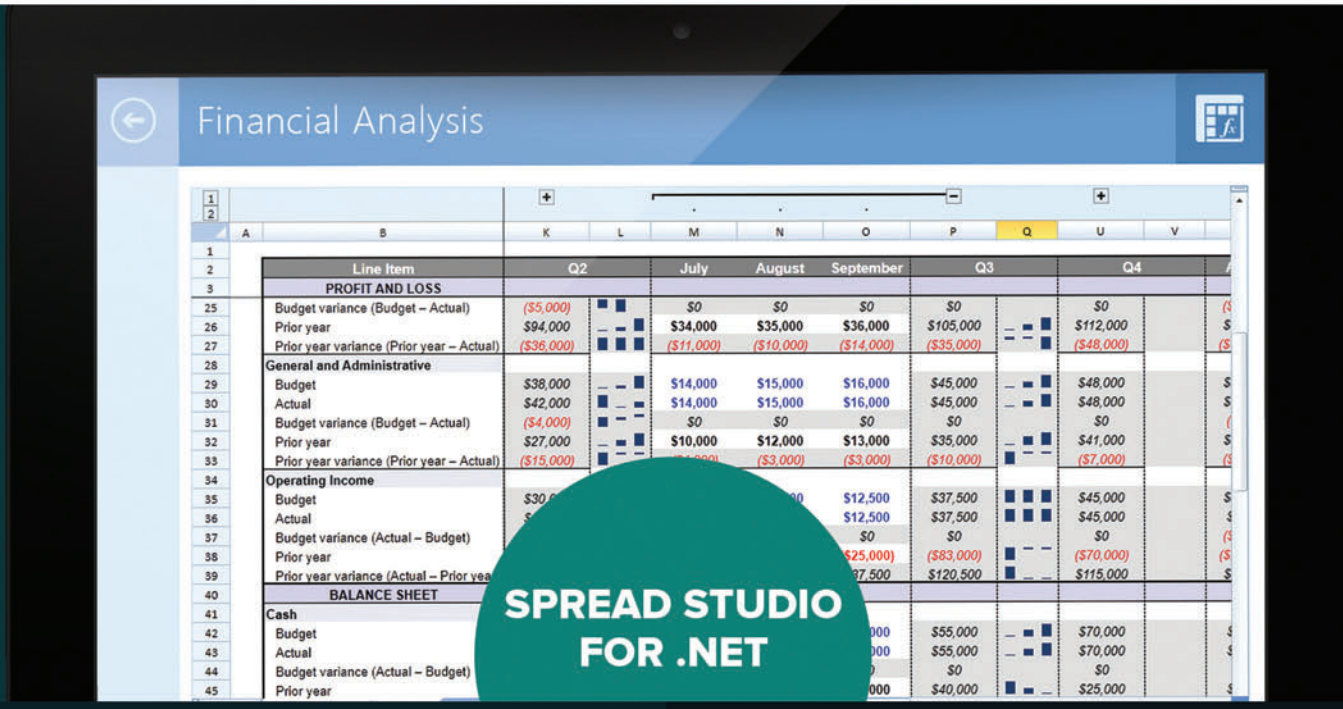
Figure 3 **An Example of How the Ability to Substitute Might Be Broken**

```
class GrooveControl: Control {
  protected bool isMoonWalking;
  protected override void OnMouseDown(MouseButtonEventArgs e) {
    isMoonWalking = CaptureMouse();
    base.OnMouseDown(e);
  }

  protected override void OnMouseUp(MouseButtonEventArgs e) {
    base.OnMouseUp(e);
    if (isMoonWalking) {
      ReleaseMouseCapture();
      isMoonWalking = false;
    }
  }
}
```

THE EXPERTS IN SPREADSHEETS

When you need the power of a spreadsheet combined with the functionally of an advanced data grid, you need Spread, the world's #1 selling spreadsheet component for Microsoft Visual Studio development.



Spread provides a flexible and familiar spreadsheet/grid architecture, advanced charting, and a powerful formula library that is ideal for creating financial modeling and risk analysis, budgeting, insurance, scientific, and many other applications.

AVAILABLE FOR:

Windows Forms · ASP.NET · WPF · Silverlight · WinRT

- Read Microsoft Excel files and/or generate Excel output from your business application
- Provide Excel like characteristics, such as complete formula calculation support, enhanced filtering, sorting, conditional formatting, cell types, sheets, and more
- Create Complex Form/Data-entry Layouts that include many fields and calculations, such as insurance forms or tax forms
- Add Data Visualization, Analysis, and Dashboards

ComponentOne
a division of GrapeCity®

DOWNLOAD YOUR FREE TRIAL

▶ componentone.com

software. They typically read in a large log file or database table for analysis. Modifying the data was never part of the agenda.

Admittedly, the IEnumerable interface was intended to be the minimal, read-only interface. With the addition of LINQ extension methods, it has started to fulfill that destiny. Microsoft has also recognized the gap in indexable collection interfaces. The company has addressed this in the 4.5 version of the .NET Framework with the addition of IReadOnlyList<T>, now implemented by many framework collections.

You'll remember these beauties in the old ICollection interface:

```
public interface ICollection : IEnumerable {
  ...
  object SyncRoot { get; }
  bool IsSynchronized { get; }
  ...
}
```

In other words, before you can iterate the collection, you must first potentially lock on its SyncRoot. A number of inheritors even implemented those particular items explicitly just to help hide their shame at having to implement them. The expectation in multi-threaded scenarios became that you lock on the collection everywhere you use it (rather than using the SyncRoot).

Most of you want to encapsulate your collections so they can be accessed in a thread-safe fashion. Instead of using foreach, you must encapsulate the multi-threaded data store and only expose a ForEach method that takes a delegate instead. Fortunately, newer collection classes such as the concurrent collections in the .NET Framework 4 or the immutable collections now available for the .NET Framework 4.5 (through NuGet) have eliminated much of this madness.

The .NET Stream abstraction shares the same faults of being way too large, including both readable and writable elements and synchronization flags. However, it does include properties to determine the writability: CanRead, CanWrite, CanSeek and so on. Compare if (stream.CanWrite) to if (stream is IWritableStream). For those of you creating streams that aren't writable, the latter is certainly appreciated.

Now, look at the code in **Figure 4**.

What's the problem here? Your service initialization and cleanup should come through one of the fantastic inversion of control (IoC) containers commonly available for the .NET Framework, instead of being reinvented. For example's sake, nobody cares about Initialization and Cleanup other than the service manager/container/boostrapper—whatever code loads up these services. That's the code that cares. You don't want anyone else calling Cleanup prematurely. C# has a mechanism called explicit implementation for helping with this. You can implement the service more cleanly like this:

```
public class SocialIntroductionsService: INerdService
{
  Type[] INerdService.Dependencies { get { return Type.EmptyTypes; } }

  void INerdService.Initialize(IEnumerable<INerdService> dependencies)
  { ... }

  void INerdService.Cleanup() {  ... }

...
}
```

Generally, you want to design your interfaces with some purpose other than pure abstraction of a single concrete class. This gives you the means to organize and extend. However, there are at least two notable exceptions.

First, interfaces tend to change less often than their concrete implementations. You can use this to your advantage. Put the interfaces in a separate assembly. Let the consumers reference only the interface assembly. It helps compilation speed. It helps you avoid putting properties on the interface that don't belong (because inappropriate property types aren't available with a proper project hierarchy). If corresponding abstractions and interfaces are in the same file, something has gone wrong. Interfaces fit in the project hierarchy as parents of their implementations and peers of the services (or abstractions of the services) that use them.

Second, by definition, interfaces don't have any dependencies. Hence, they lend themselves to easy unit testing through object mocking/proxy frameworks. That brings me to the next and final principle.

The Dependency Inversion Principle

Dependency Inversion means to depend on abstractions instead of concrete types. There's a lot of overlap between this principle and the others already discussed. Many of the previous examples include a failure to depend on abstractions.

In his book, "Domain Driven Design" (Addison-Wesley Professional, 2003), Eric Evans outlines some object classifications that are useful in discussing Dependency Inversion. To summarize the book, it's useful to classify your object into one of these three groups: values, entities or services.

Values refer to objects with no dependencies that are typically transient and immutable. They're generally not abstracted and you can instantiate them at will. However, there's nothing wrong with abstracting them, especially if you can get all the benefits of abstractions. Some values might grow into entities over time. Entities are your business Models and ViewModels. They're built from value types and other entities. It's useful to have abstractions for these items, especially if you have one ViewModel that represents several different variants of a Model or vice versa. Services are the classes that contain, organize, service and use the entities.

With this classification in mind, Dependency Inversion deals primarily with services and the objects that need them. Service-specific methods should always be captured in an interface. Wherever you need to access that service, you access it via the interface. Don't use a concrete service type in your code anywhere other than where the service is constructed.

Figure 4 **An Example of Unnecessary Initialization and Cleanup**

```
// Up a level in the project hierarchy
public interface INerdService {
  Type[] Dependencies { get; }
  void Initialize(IEnumerable<INerdService> dependencies);
  void Cleanup();
}

public class SocialIntroductionsService: INerdService
{
  public Type[] Dependencies { get { return Type.EmptyTypes; } }
  public void Initialize(IEnumerable<INerdService> dependencies)
  { ... }

  public void Cleanup() { ... }

  ...
}
```

Services generally depend on other services. Some ViewModels depend on services, especially container and factory-type services. Therefore, services are generally difficult to instantiate for testing because you need the full service tree. Abstract their essence into an interface. Then all references to services should be made through that interface so they can be easily mocked up for testing purposes.

You can create abstractions at any level in the code. When you find yourself thinking, "Wow, it's going to be painful for A to support B's interface and B to support A's interface," that's the perfect time to introduce a new abstraction in the middle. Make usable interfaces and rely on them.

The adapter and mediator patterns can help you conform to the preferred interface. It sounds like extra abstractions bring extra code, but generally that's not true. Taking partial steps toward interoperability helps you organize code that would've had to exist for A and B to talk to each other anyway.

Years ago, I read that a developer should "always reuse code." It seemed too simple at the time. I couldn't believe such a simple mantra could penetrate the spaghetti all over my screen. Over time, though, I've learned. Look at the code here:

```
private readonly IRamenContainer _ramenContainer; // A dependency
public bool Recharge()
{
  if (_ramenContainer != null)
  {
    var toBeConsumed = _ramenContainer.Prepare();
    return Consume(toBeConsumed);
  }
  return false;
}
```

Do you see any repeated code? There's the double read on _ramenContainer. Technically speaking, the compiler will eliminate this with an optimization called "common sub-expression elimination." For discussion, suppose you were running in a multi-threaded situation and the compiler actually repeated class field reads in the method. You would run the risk that your class variable is changed to null before it's even used.

How do you fix this? Introduce a local reference above the if statement. This rearrangement requires you to add a new item at or above the outer scope. The principle is the same in your project organization! When you reuse code or abstractions, you eventually arrive at a useful scope in your project hierarchy. Let the dependencies drive the inter-project reference hierarchy.

Now, look at this code:

```
public IList<Nerd> RestoreNerds(string filename)
{
  if (File.Exists(filename))
  {
    var serializer = new XmlSerializer(typeof(List<Nerd>));
    using (var reader = new XmlTextReader(filename))
      return (List<Nerd>)serializer.Deserialize(reader);
  }
  return null;
}
```

Is it depending on abstractions?

No, it isn't. It begins with a static reference to the file system. It's using a hardcoded deserializer with hardcoded type references. It expects exception handling to occur outside the class. This code is impossible to test without the accompanying storage code.

Typically, you would move this into two abstractions: one for the storage format and one for the storage medium. Some examples of storage formats include XML, JSON and Protobuf binary data. Storage mediums include direct files on a disk and databases. A third abstraction is also typical in this type of system: some kind of rarely changing memento representing the object to be stored.

Consider this example:

```
class MonsterCardCollection
{
  private readonly IMsSqlDatabase _storage;
  public MonsterCardCollection(IMsSqlDatabase storage)
  {
    _storage = storage;
  }
  ...
}
```

Can you see anything wrong with these dependencies? The clue is in the dependency name. It's platform-specific. The service isn't platform-specific (or at least it's attempting to avoid a platform dependency by using an external storage engine). This is a situation where you need to employ the adapter pattern.

> Years ago, I read that a developer should "always reuse code." It seemed too simple at the time.

When dependencies are platform-specific, the dependents will end up with their own platform-specific code. You can avoid this with one additional layer. The additional layer will help you organize the projects in such a way that the platform-specific implementation exists in its own special project (with all its platform-specific references). You'll only need to reference the project containing all the platform-specific code by the start-up application project. Platform wrappers tend to be large; don't duplicate them more than necessary.

Dependency Inversion brings together the entire set of principles discussed in this article. It uses clean, purposeful abstractions you can fill with concrete implementations that don't break the underlying service state. That's the goal.

Indeed, the SOLID principles are generally overlapping in their effects upon sustainable computer code. The vast world of intermediate (meaning easily decompiled) code is fantastic in its ability to reveal the full extent to which you may extend any object. A number of .NET library projects fade over time. That's not because the idea was faulty; they just couldn't safely extend into the unanticipated and varying needs of the future. Take pride in your code. Apply the SOLID principles and you'll see your code's lifespan increase. ■

Brannon B. King *has worked as a full-time software developer for 12 years, eight of which have been spent deep in C# and the .NET Framework. His most recent work has been with Autonomous Solutions Inc. (ASI) near Logan, Utah (asirobots.com). ASI is unique in its ability to foster a contagious love of C#; the crew at ASI takes passion in fully utilizing the language and pushing the .NET Framework to its limits. Reach him at countprimes@gmail.com.*

Thanks to the following technical experts for reviewing this article:
Max Barfuss (ASI) and Brian Pepin (Microsoft)

YOUR GUIDE TO THE .NET DEVELOPMENT UNIVERSE

From August 18 – 22, 2014, developers, engineers, software architects and designers will land in Redmond, WA at the idyllic Microsoft headquarters for 5 days of cutting-edge education on the Microsoft Platform.

Come experience code at the source – rub elbows with Microsoft stars, get the inside scoop on what's next, and learn what you need to know now. Over 5 days and 60+sessions and workshops, you'll explore the .NET Development Universe, receiving the kind of practical, unbiased training you can only get at Visual Studio Live!

Tracks Include:

- Visual Studio/.NET Framework
- Windows Client
- JavaScript/HTML5 Client
- ASP.NET
- Cloud Computing
- Windows Phone
- Cross-Platform Mobile Development
- SharePoint
- SQL Server

PRODUCED BY

1105 MEDIA

REDMOND 2014

August 18 – 22
Microsoft Campus
Redmond, WA

Register by June 11 and Save $400!

Use Promo Code VSLMAY4

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

vslive.com/redmond

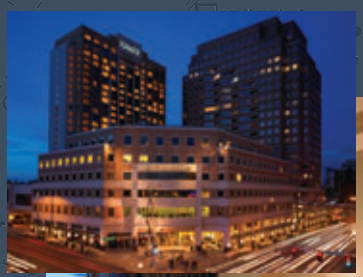
TURN THE PAGE FOR MORE EVENT DETAILS

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

MICROSOFT HEADQUARTERS
REDMOND 2014

Visual Studio Live! has partnered with the Hyatt Regency Bellevue for conference attendees at a special reduced rate.



CONNECT WITH VISUAL STUDIO LIVE!

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

linkedin.com – Join the "Visual Studio Live" group!

Register at vslive.com/redmond

Use Promo Code VSLMAY4

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

AGENDA AT-A-GLANCE

Visual Studio / .NET Framework | **Windows Client**

START TIME	END TIME	
7:00 AM	8:00 AM	
8:00 AM	12:00 PM	**MW01** - Workshop: Modern UX Design - *Billy Hollis*
12:00 PM	2:30 PM	
2:30 PM	6:00 PM	**MW01** - Workshop: Modern UX Design - *Billy Hollis*
7:00 PM	9:00 PM	

START TIME	END TIME	
7:30 AM	8:30 AM	
8:30 AM	9:30 AM	
9:45 AM	11:00 AM	**T01** - What's New in WinRT Development - *Rockford Lhotka*
11:15 AM	12:30 PM	**T06** - What's New for XAML Windows Store Apps - *Ben Dewey*
12:30 PM	2:30 PM	
2:30 PM	3:45 PM	**T11** - Interaction Design Principles and Patterns - *Billy Hollis*
3:45 PM	4:15 PM	
4:15 PM	5:30 PM	**T16** - Applying UX Design in XAML - *Billy Hollis*
5:30 PM	7:00 PM	

START TIME	END TIME	
7:30 AM	8:00 AM	
8:00 AM	9:00 AM	
9:15 AM	10:30 AM	**W01** - Getting Started with Windows Phone Development - *Nick Landry*
10:45 AM	12:00 PM	**W06** - Build Your First Mobile App in 1 hour with Microsoft App Studio - *Nick Landry*
12:00 PM	1:30 PM	
1:30 PM	2:45 PM	**W11** - What's New for HTML/WinJS Windows Store Apps - *Ben Dewey*
2:45 PM	3:15 PM	
3:15 PM	4:30 PM	**W16** - Windows 8 HTML/JS Apps for the ASP.NET Developer - *Adam Tuliper*
8:00 PM	10:00 PM	

START TIME	END TIME	
7:30 AM	8:00 AM	
8:00 AM	9:15 AM	**TH01** - Developing Awesome 3D Applications with Unity and C#/JavaScript - *Adam Tuliper*
9:30 AM	10:45 AM	**TH06** - What's New in WPF 4.5 - *Walt Ritscher*
11:00 AM	12:15 PM	**TH11** - Implementing M-V-VM (Model-View-View Model) for WPF - *Philip Japikse*
12:15 PM	2:15 PM	
2:15 PM	3:30 PM	**TH16** - Build Maintainable Windows Store Apps with MVVM and Prism - *Brian Noyes*
3:45 PM	5:00 PM	**TH21** - Make Your App Alive with Tiles and Notifications - *Ben Dewey*

START TIME	END TIME	
7:30 AM	8:00 AM	
8:00 AM	12:00 PM	**WF01** - Workshop: Service Orientation
12:00 PM	1:00 PM	
1:00 PM	5:00 PM	**WF01** - Workshop: Service Orientation

Speakers and sessions subject to change

August 18 – 22 | Microsoft Campus | Redmond, WA

Cloud Computing	Windows Phone	Cross-Platform Mobile Development	ASP.NET	JavaScript / HTML5 Client	SharePoint	SQL Server

Visual Studio Live! Pre-Conference Workshops: Monday, August 18, 2014 (Separate entry fee required)

Pre-Conference Workshop Registration - Coffee and Morning Pastries	
MW02 - Workshop: Data-Centric Single Page Applications with Durandal, Knockout, Breeze, and Web API - *Brian Noyes*	**MW03** - Workshop: SQL Server for Developers - *Andrew Brust & Leonard Lobel*
Lunch @ The Mixer - Visit the Microsoft Company Store & Visitor Center	
MW02 - Workshop: Data-Centric Single Page Applications with Durandal, Knockout, Breeze, and Web API - *Brian Noyes*	**MW03** - Workshop: SQL Server for Developers - *Andrew Brust & Leonard Lobel*
Dine-A-Round Dinner	

Visual Studio Live! Day 1: Tuesday, August 19, 2014

Registration - Coffee and Morning Pastries			
Keynote: To Be Announced			
T02 - Creating Data-Driven Mobile Web Apps with ASPNET MVC and jQuery Mobile - *Rachel Appel*	**T03** - HTML5 for Better Web Sites - *Robert Boedigheimer*	**T04** - Introduction to Windows Azure - *Vishwas Lele*	**T05** - What's New in the Visual Studio 2013 IDE
T07 - To Be Announced	**T08** - Great User Experiences with CSS 3 - *Robert Boedigheimer*	**T09** - Windows Azure Cloud Services - *Vishwas Lele*	**T10** - What's New for Web Developers in Visual Studio this Year? - *Mads Kristensen*
Lunch - Visit Exhibitors			
T12 - Getting Started with Xamarin - *Walt Ritscher*	**T13** - Building Real Time Applications with ASP.NET SignalR - *Rachel Appel*	**T14** - To Be Announced	**T15** - Why Browser Link Changes Things, and How You Can Write Extensions? - *Mads Kristensen*
Sponsored Break - Visit Exhibitors			
T17 - Building Multi-Platform Mobile Apps with Push Notifications - *Nick Landry*	**T18** - JavaScript for the C# Developer - *Philip Japikse*	**T19** - Windows Azure SQL Database – SQL Server in the Cloud - *Leonard Lobel*	**T20** - Katana, OWIN, and Other Awesome Codenames: What's coming? - *Howard Dierking*
Microsoft Ask the Experts & Exhibitor Reception – Attend Exhibitor Demos			

Visual Studio Live! Day 2: Wednesday, August 20, 2014

Registration - Coffee and Morning Pastries			
Keynote: To Be Announced			
W02 - Learning Entity Framework 6 - *Leonard Lobel*	**W03** - Build Data-Centric HTML5 Single Page Applications with Breeze - *Brian Noyes*	**W04** - To Be Announced	**W05** - Upgrading Your Existing ASP.NET Apps - *Pranav Rastogi*
W07 - Programming the T-SQL Enhancements in SQL Server 2012 - *Leonard Lobel*	**W08** - Knocking it Out of the Park, with Knockout.JS - *Miguel Castro*	**W09** - Solving Security and Compliance Challenges with Hybrid Clouds - *Eric D. Boyd*	**W10** - To Be Announced
Birds-of-a-Feather Lunch - Visit Exhibitors			
W12 - SQL Server 2014: Features Drill-down - *Scott Klein*	**W13** - JavaScript: Turtles, All the Way Down - *Ted Neward*	**W14** - Zero to Connected with Windows Azure Mobile Services - *Brian Noyes*	**W15** - To Be Announced
Sponsored Break - Exhibitor Raffle @ 2:55 pm (Must be present to win)			
W17 - SQL Server 2014 In-memory OLTP - Deep Dive - *Scott Klein*	**W18** - AngularJS JumpStart - *Brian Noyes*	**W19** - Leveraging Azure Web Sites - *Rockford Lhotka*	**W20** - Cool ALM Features in Visual Studio 2013 - *Brian Randell*
Lucky Strike Evening Out Party			

Visual Studio Live! Day 3: Thursday, August 21, 2014

Registration - Coffee and Morning Pastries			
TH02 - AWS for the SQL Server Professional - *Lynn Langit*	**TH03** - Sexy Extensibility Patterns - *Miguel Castro*	**TH04** - Beyond Hello World: A Practical Introduction to Node.js on Windows Azure Websites - *Rick Garibay*	**TH05** - Leveraging Visual Studio Online - *Brian Randell*
TH07 - Real-world Predictive Analytics with PowerBI and Predixion Software - *Lynn Langit*	**TH08** - What's New in MVC 5 - *Miguel Castro*	**TH09** - From the Internet of Things to Intelligent Systems: A Developer's Primer - *Rick Garibay*	**TH10** - Essential C# 6.0 - *Mark Michaelis*
TH12 - Excel, Power BI and You: An Analytics Superhub - *Andrew Brust*	**TH13** - What's New in Web API 2 - *Miguel Castro*	**TH14** - Building Mobile Applications with SharePoint 2013 - *Darrin Bishop*	**TH15** - Performance and Diagnostics Hub in Visual Studio 2013 - *Brian Peek*
Lunch @ The Mixer - Visit the Microsoft Company Store & Visitor Center			
TH17 - Big Data 101 with HDInsight - *Andrew Brust*	**TH18** - Finding and Consuming Public Data APIs - G. *Andrew Duthie*	**TH19** - Building Apps for SharePoint - *Mark Michaelis*	**TH20** - Git for the Microsoft Developer - *Eric D. Boyd*
TH22 - NoSQL for the SQL Guy - *Ted Neward*	**TH23** - Provide Value to Customers and Enhance Site Stickiness By Creating an API - *G. Andrew Duthie*	**TH24** - Data as Information: Understanding Your SharePoint 2013 Business Intelligence Options - *Darrin Bishop*	**TH25** - Writing Asynchronous Code Using .NET 4.5 and C# 5.0 - *Brian Peek*

Visual Studio Live! Post-Conference Workshops: Friday, August 22, 2014 (Separate entry fee required)

Post-Conference Workshop Registration - Coffee and Morning Pastries	
Technologies: Designing, Developing, & Implementing WCF and the Web API - *Miguel Castro*	**WF02** - Workshop: A Day of Windows Azure - *Eric D. Boyd*
Lunch	
Technologies: Designing, Developing, & Implementing WCF and the Web API - *Miguel Castro*	**WF02** - Workshop: A Day of Windows Azure - *Eric D. Boyd*

JAMES MCCAFFREY

Association Rule Learning

Association rule learning is a technique from the field of machine learning that extracts if-then rules from a set of data. For example, if the data being explored is a set of supermarket transactions, one association rule might be, "IF a customer buys apples and coffee THEN there is a high likelihood he will also buy butter and donuts."

There are several types of association rules. This article explains how to extract high-confidence rules, which are characterized by being true in a specified minimum percentage of the transactions being analyzed. Association rule learning can be applied to many kinds of data besides purchasing transactions, including system log files, user search queries and natural UI commands. This article explains how high-confidence association rule learning works and presents a complete demo program.

The best way to see where this article is headed is to take a look at the demo program in **Figure 1**. The demo sets up 10 dummy transactions, encoded so each item is a 0-based value. For example, the first transaction is (0 3 4 11), which means that items 0, 3, 4 and 11 occur together. In most cases, duplicate transactions are allowed, such as transactions 2 and 3.

> Each candidate rule is evaluated to determine if the rule meets a user-supplied minimum threshold of likelihood called the confidence value.

Although you can perform a search for high-confidence association rules directly on the encoded transactions data, in most situations frequent item-sets are first extracted from the transactions. An item-set is a subset of all possible transactions, and frequent item-sets are those that meet some user-specified minimum number of occurrences (called the support level) in the transactions set. In the demo, using a support value of 0.30 (so an item-set must occur in at least 0.30 * 10 = 3 transactions), there are eight frequent item-sets. The first is (2 5). Items 2 and 5 occur together in three transactions: 7, 8 and 9. Frequent item-sets eliminate outlier transactions that might generate a very high-confidence rule but are so rare the rule isn't particularly relevant. In most cases, frequent item-sets are distinct, with no duplicates allowed. Extracting frequent item-sets from transactions is a surprisingly difficult task. See "Frequent Item-Sets for Association Rule Learning" in the January 2013 issue of *MSDN Magazine* at msdn.microsoft.com/magazine/dn519928.

Behind the scenes, the demo uses the list of frequent item-sets to generate candidate rules. Each candidate rule is evaluated to determine if the rule meets a user-supplied minimum threshold

```
file:///C:/AssocRuleLearn/bin/Debug/AssocRuleLearn.EXE
Begin high-confidence association rule demo

Encoded transactions are:
 0: ( 0 3 4 11 )
 1: ( 1 4 5 )
 2: ( 3 4 6 7 )
 3: ( 3 4 6 7 )
 4: ( 0 5 )
 5: ( 3 5 9 )
 6: ( 2 3 4 7 )
 7: ( 2 5 8 )
 8: ( 0 1 2 5 10 )
 9: ( 2 3 5 6 7 9 )

Frequent item-sets (support = 0.30) are:
 0: ( 2 5 )
 1: ( 3 4 )
 2: ( 3 6 )
 3: ( 3 7 )
 4: ( 4 7 )
 5: ( 6 7 )
 6: ( 3 4 7 )
 7: ( 3 6 7 )

Setting minimum confidence =  0.700

Starting high-confidence rule extraction

Done. Extracted association rules are:

IF ( 2 )      THEN ( 5 )      conf = 0.75
IF ( 4 )      THEN ( 3 )      conf = 0.80
IF ( 6 )      THEN ( 3 )      conf = 1.00
IF ( 7 )      THEN ( 3 )      conf = 1.00
IF ( 7 )      THEN ( 4 )      conf = 0.75
IF ( 6 )      THEN ( 7 )      conf = 1.00
IF ( 7 )      THEN ( 6 )      conf = 0.75
IF ( 7 )      THEN ( 3 4 )    conf = 0.75
IF ( 3 4 )    THEN ( 7 )      conf = 0.75
IF ( 3 7 )    THEN ( 4 )      conf = 0.75
IF ( 4 7 )    THEN ( 3 )      conf = 1.00
IF ( 6 )      THEN ( 3 7 )    conf = 1.00
IF ( 7 )      THEN ( 3 6 )    conf = 0.75
IF ( 3 6 )    THEN ( 7 )      conf = 1.00
IF ( 3 7 )    THEN ( 6 )      conf = 0.75
IF ( 6 7 )    THEN ( 3 )      conf = 1.00

End high-confidence association rule demo
```

Figure 1 **Finding High-Confidence Association Rules**

Code download available at msdn.microsoft.com/magazine/msdnmag0514.

of likelihood called the confidence value. Those candidate rules that meet or exceed the likelihood level are identified as high-confidence rules. In the demo, the confidence value is set to 0.700, which means that a candidate rule must be true for at least 70 percent of the transactions for which the rule is applicable.

In the demo, 16 high-confidence rules were found. The first rule listed is IF (2) THEN (5), with a computed 0.75 confidence value. You can think of this as, "If a transaction has item 2, then the transaction probably also contains item 5." The IF part of the rule, which is called the antecedent, applies to four transactions: 6, 7, 8 and 9. The rule is true for three of those four transactions: 7, 8 and 9, so the computed confidence is 3/4 = 0.75.

> Those candidate rules that meet or exceed the likelihood level are identified as high-confidence rules.

Notice that high-confidence rules are not necessarily symmetric. There is no IF (5) THEN (2) rule because that rule is applicable to transactions 1, 4, 5, 7, 8, and 9, but is true only for transactions 7, 8, and 9, so the confidence value of 3/6 = 0.50 doesn't meet the minimum 0.700 value.

This article assumes you have advanced programming skills but doesn't assume you know anything about association rule learning. The demo program is coded using C# but you should be able to refactor the code to other .NET languages such as Visual Basic or IronPython without too much difficulty. Most normal error-checking has been removed to keep the size of the code small and the main ideas clear. The complete source code for the demo is presented in this article and the code is also available as a download from msdn.microsoft.com/magazine/msdnmag0514.

The Rule-Finding Algorithm

The algorithm used by the demo program to find high-confidence rules is illustrated in **Figure 2**. The diagram shows how frequent item-set (3 4 7) generates two non-high-confidence rules and four high-confidence rules. The algorithm starts by generating all mathematical combinations for size k = 1 through size item-set length - 1. Mathematical combinations are the key to the association rule learning algorithm presented in this article. A mathematical combination is a set of numbers that represents a subset. For example, if there are 5 items = (0, 1, 2, 3, 4) and the subset size k is 3, the 10 possible combination elements are:

```
(0, 1, 2)
(0, 1, 3)
(0, 1, 4)
(0, 2, 3)
(0, 2, 4)
(0, 3, 4)
(1, 2, 3)
(1, 2, 4)
(1, 3, 4)
(2, 3, 4)
```

The elements in a mathematical combination are not transaction items, they're just numbers. In **Figure 2**, each combination is applied to the frequent item-set to generate a subset of the item-set, which is interpreted as the antecedent (if-part) of a candidate rule. For example, the last combination for k = 2 is (1, 2) so the items in the frequent item-set (3 4 7) at indices 1 and 2 are used as the candidate rule antecedent: "IF (4 7)." The then-part of the candidate rule consists of those items in the item-set being examined that are not used in the if-part. So for item-set (3 4 7), if the antecedent is (4 7), the then-part (called the consequent) is (3), and the full candidate rule is "IF (4 7) THEN (3)."

Somewhat surprisingly, the then-part of a candidate rule isn't needed to compute the rule's confidence value. Consider the candidate rule IF (4 7) THEN (3). To compute the confidence, a count of transactions to which the rule is applicable is needed. These would be those transactions that contain items 4 and 7, which in the case of the demo is 3 (transactions 2, 3 and 6). The second part of the computation needed is the number of applicable transactions for which the candidate rule is true, in other words, those transactions that have items 4, 7 and also item 3. But this is just the number of transactions that contain the source frequent item-set (3 4 7).

In **Figure 2**, the confidence values for each of the six candidate rules are computed, and the four that meet the confidence threshold are identified as high-confidence rules.

To summarize, starting from a set of frequent item-sets that meet a minimum frequency-of-occurrence support level, for each frequent item, all mathematical combinations from size 1 through one less than the number of items in the item-set are generated. Each combination determines an IF part and a THEN part of a candidate rule. Candidate rules that meet a minimum confidence level, which is the proportion of applicable frequent item-sets for which the rule is true, are labeled as good (high-confidence) rules and saved.

Overall Program Structure

To create the demo program I launched Visual Studio. The demo program has no significant .NET dependencies and any version of Visual Studio that supports the Microsoft .NET Framework 2.0 or later should work fine. I created a C# console application program and named it AssocRuleLearn. After the template-generated code

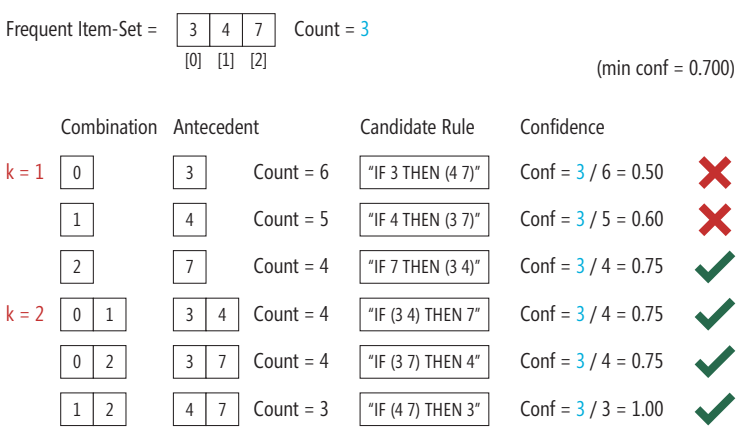


Figure 2 **Algorithm to Find High-Confidence Rules**

Figure 3 **Association Rule Demo Program Structure**

```
using System;
using System.Collections.Generic;
namespace AssocRuleLearn
{
  class AssocRuleProgram
  {
    static void Main(string[] args)
    {
      try
      {
        Console.WriteLine("\nBegin demo\n");

        List<int[]> transactions = new List<int[]>();
        transactions.Add(new int[] { 0, 3, 4, 11 });
        transactions.Add(new int[] { 1, 4, 5 });
        transactions.Add(new int[] { 3, 4, 6, 7 });
        transactions.Add(new int[] { 3, 4, 6, 7 });
        transactions.Add(new int[] { 0, 5 });
        transactions.Add(new int[] { 3, 5, 9 });
        transactions.Add(new int[] { 2, 3, 4, 7 });
        transactions.Add(new int[] { 2, 5, 8 });
        transactions.Add(new int[] { 0, 1, 2, 5, 10 });
        transactions.Add(new int[] { 2, 3, 5, 6, 7, 9 });

        ShowList(transactions);

        List<int[]> freqItemSets = new List<int[]>();
        freqItemSets.Add(new int[] { 2, 5 });
        freqItemSets.Add(new int[] { 3, 4 });
        freqItemSets.Add(new int[] { 3, 6 });
        freqItemSets.Add(new int[] { 3, 7 });
        freqItemSets.Add(new int[] { 4, 7 });
        freqItemSets.Add(new int[] { 6, 7 });
        freqItemSets.Add(new int[] { 3, 4, 7 });
        freqItemSets.Add(new int[] { 3, 6, 7 });

        ShowList(freqItemSets);

        double minConPct = 0.70;
        List<Rule> goodRules =
          GetHighConfRules(freqItemSets, transactions, minConPct);

        Console.WriteLine("\nDone. Rules are:\n");
        for (int i = 0; i < goodRules.Count; ++i)
          Console.WriteLine(goodRules[i].ToString());

        Console.WriteLine("\nEnd demo\n");
        Console.ReadLine();
      }
      catch (Exception ex)
      {
        Console.WriteLine(ex.Message);
        Console.ReadLine();
      }
    } // Main

    static void ShowList(List<int[]> trans)
    {
      for (int i = 0; i < trans.Count; ++i)
      {
        Console.Write(i.ToString().PadLeft(2) + ": ( ");
        for (int j = 0; j < trans[i].Length; ++j)
          Console.Write(trans[i][j] + " ");
        Console.WriteLine(")");
      }
    }

    static List<Rule> GetHighConfRules(List<int[]> freqItemSets,
      List<int[]> transactions, double minConfidencePct) { . . }

    static int[] NewCombination(int k) { . . }
    static int[] NextCombination(int[] comb, int n) { . . }
    static int[] MakeAntecedent(int[] itemSet, int[]comb) { . . }
    static int[] MakeConsequent(int[] itemSet, int[]comb) { . . }

    static int CountInTrans(int[] itemSet,
      List<int[]> trans, Dictionary<int[], int> countDict) { . . }
    static bool IsSubsetOf(int[] itemSet, int[] trans) { . . }
    static int IndexOf(int[] array, int item, int startIdx) { . . }
  }

  public class Rule
  {
    public int[] antecedent; // If part
    public int[] consequent; // Then part
    public double confidence;

    public Rule(int[] antecedent, int[] consequent, double confidence)
    {
      this.antecedent = new int[antecedent.Length];
      Array.Copy(antecedent, this.antecedent, antecedent.Length);
      this.consequent = new int[consequent.Length];
      Array.Copy(consequent, this.consequent, consequent.Length);
      this.confidence = confidence;
    }

    public override string ToString()
    {
      string s = "IF ( ";
      for (int i = 0; i < antecedent.Length; ++i)
        s += antecedent[i] + " ";
      s += ")";
      s = s.PadRight(13);

      string t = " THEN ( ";
      for (int i = 0; i < consequent.Length; ++i)
        t += consequent[i] + " ";
      t += ") ";
      t = t.PadRight(17);

      return s + t + "conf = " + confidence.ToString("F2");
    }
  }
}
```

loaded into the editor, in the Solution Explorer window I renamed Program.cs to the more descriptive AssocRuleProgram.cs and Visual Studio automatically renamed class Program for me. At the top of the source code, I deleted all references to unneeded namespaces, leaving just those to System and Collections.Generic.

The overall program structure, with some WriteLine statements removed and a few minor edits to save space, is presented in **Figure 3**.

The demo defines both transactions and item-sets as arrays of type int and sets up a List of hardcoded transactions. You may want to create a program-defined ItemSet class. In most situations the raw transaction data will be in a text file or SQL database and need to be encoded. The demo sets up hardcoded frequent item-sets. In all but very small demo scenarios you'll need to extract frequent item-sets programmatically, which is a very difficult task.

All the work of finding high-confidence rules is performed by method GetHighConfRules. That method requires a user-specified minimum confidence percentage parameter value. Meaningful confidence values will vary from problem to problem.

The demo program uses a program-defined Rule class. Class members are the rule antecedent (if-part), rule consequent (then-part) and the computed confidence value. The class has just a constructor and a ToString method, which is hacked to format the demo data nicely.

Method GetHighConfRules calls several helper methods. Method CountInTrans counts the number of times an item-set or rule antecedent or rule consequent occurs in a list of transactions. This helper method calls sub-helper IsSubsetOf, which in turn calls sub-helper IndexOf. Method NewCombination creates a

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

Figure 4 **Pseudocode for Method GetHighConfRules**

```
for each frequent item-set
  ctItemSet = count times item-set is in transactions
  for subset length 1 to item-set length - 1
    create a new math combination
    loop over each possible math combination
      create candidate rule if-part
      create candidate rule then-part
      compute confidence of candidate rule
      if candidate rule meets min confidence, save
    end loop
  end for
end for
return saved rules
```

mathematical combination. Method NextCombination returns the next combination element for a given combination. Methods MakeAntecedent and MakeConsequent return the if-part and the then-part for a candidate rule, given a frequent item-set and a mathematical combination.

> Somewhat surprisingly, the then-part of a candidate rule isn't needed to compute the rule's confidence value.

Mathematical Combinations

If you refer to the diagram in **Figure 2**, you'll see the algorithm used by the demo program requires creating mathematical combinations and the ability to generate the successor to a given combination. Method NewCombination is defined as:

```
static int[] NewCombination(int k)
{
  int[] result = new int[k];
  for (int i = 0; i < result.Length; ++i)
    result[i] = i;
  return result;
}
```

Here, a combination is an array of int. The input parameter k determines the size of the combination. A new combination is values 0 through k-1, in order. Method NextCombination is defined as:

```
static int[] NextCombination(int[] comb, int n)
{
  int[] result = new int[comb.Length];
  int k = comb.Length;
  if (comb[0] == n - k) return null;
  Array.Copy(comb, result, comb.Length);
  int i = k - 1;
  while (i > 0 && result[i] == n - k + i) --i;
  ++result[i];
  for (int j = i; j < k - 1; ++j)
    result[j + 1] = result[j] + 1;
  return result;
}
```

Mathematical combinations are fascinating topics in their own right, and somewhat complicated. Method NextCombination is short but not trivial. It accepts a combination, and the number of possible items in the combination. The method returns the lexicographical successor to the input combination, or null if the input combination is the last one in lexicographical order. For example, if a combination with n = 6 and k = 3 is (1, 4, 5) then the next combination is (2, 3, 4).

Generating Rule Antecedents and Consequents

Helper method MakeAntecedent accepts a frequent item-set and a mathematical combination, and returns the if-part of a candidate rule. For example, if a frequent item-set is (1 3 4 6 8) and a combination is (0, 2), the item values at indices 0 and 2 are extracted giving an antecedent of (1 4):

```
static int[] MakeAntecedent(int[] itemSet, int[] comb)
{
  int[] result = new int[comb.Length];
  for (int i = 0; i < comb.Length; ++i) {
    int idx = comb[i];
    result[i] = itemSet[idx];
  }
  return result;
}
```

Although short, the code can be a bit confusing because integers represent combination element values, item-set item values and item-set index values. If you trace through an example or two by hand, you should see how the method works.

Method MakeConsequent generates the then-part for a candidate rule. For example, if a frequent item-set is (1 3 4 6 8) and a combination is (0, 2), the item values at those indices not equal to 0 and 2 are extracted giving a consequent of (3 6 8), as shown here:

```
static int[] MakeConsequent(int[] itemSet, int[] comb)
{
  int[] result = new int[itemSet.Length - comb.Length];
  int j = 0; // ptr into combination
  int p = 0; // ptr into result
  for (int i = 0; i < itemSet.Length; ++i) {
    if (j < comb.Length && i == comb[j])
      ++j;
    else
      result[p++] = itemSet[i];
  }
  return result;
}
```

Figure 5 **Method GetHighConfRules**

```
static List<Rule> GetHighConfRules(List<int[]> freqItemSets,
  List<int[]> trans, double minConfidencePct)
{
  List<Rule> result = new List<Rule>();
  Dictionary<int[], int> itemSetCountDict = new Dictionary<int[], int>();

  for (int i = 0; i < freqItemSets.Count; ++i)
  {
    int[] currItemSet = freqItemSets[i]; // for clarity
    int ctItemSet = CountInTrans(currItemSet, trans, itemSetCountDict);

    for (int len = 1; len <= currItemSet.Length - 1; ++len)
    {
      int[] c = NewCombination(len);
      while (c != null) // each combination makes a candidate rule
      {
        int[] ante = MakeAntecedent(currItemSet, c);
        int[] cons = MakeConsequent(currItemSet, c); // could defer

        int ctAntecendent = CountInTrans(ante, transactions,
          itemSetCountDict);
        double confidence = (ctItemSet * 1.0) / ctAntecendent;
        if (confidence >= minConfidencePct) {
          Rule r = new Rule(ante, cons, confidence);
          result.Add(r);
        }
        c = NextCombination(c, currItemSet.Length);
      } // while each combination
    } // len each possible antecedent for curr item-set
  } // i each freq item-set
  return result;
}
```

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

I designed MakeConsequent so that it accepts the same parameters as method MakeAntecedent. A somewhat simpler but asymmetric alternative is to define MakeConsequent so that it accepts an item-set and an antecedent.

Counting Occurrences in Transactions

Key helper method CountInTrans accepts an array of int that can represent a frequent item-set or a candidate rule antecedent, a list of transactions, and a Dictionary collection, and returns the number of times the item-set or antecedent occurs. The Dictionary object stores item-set and antecedent counts and is used as a lookup so those item-sets or antecedents that have already been processed don't need to be recounted:

```
static int CountInTrans(int[] itemSet, List<int[]> trans,
  Dictionary<int[], int> countDict)
{
  if (countDict.ContainsKey(itemSet) == true)
    return countDict[itemSet];

  int ct = 0;
  for (int i = 0; i < trans.Count; ++i)
    if (IsSubsetOf(itemSet, trans[i]) == true)
      ++ct;
  countDict.Add(itemSet, ct);
  return ct;
}
```

Most of the work is done by helper method IsSubsetOf. The method takes advantage of the fact that transaction items are assumed to be stored in order, which means after a particular item has been found, the search for the next item can start at the next index position:

```
static bool IsSubsetOf(int[] itemSet, int[] trans)
{
  int foundIdx = -1;
  for (int j = 0; j < itemSet.Length; ++j) {
    foundIdx = IndexOf(trans, itemSet[j], foundIdx + 1);
    if (foundIdx == -1) return false;
  }
  return true;
}
```

Helper method IndexOf also takes advantage of the ordered property of transactions to early-exit when at an index that's past the point where a target item could possibly be:

```
static int IndexOf(int[] array, int item, int startIdx)
{
  for (int i = startIdx; i < array.Length; ++i) {
    if (i > item) return -1;
    if (array[i] == item) return i;
  }
  return -1;
}
```

Generating High-Confidence Rules

With all the helper methods in place, method GetHighConfRules can be defined without too much difficulty. **Figure 4** shows the method in high-level pseudocode.

The implementation of method GetHighConfRules is listed in **Figure 5**. Although you can use method GetHighConfRules as listed in **Figure 5**, there are many opportunities to enhance performance, usually at the expense of memory or code clarity. For example, if you refer to **Figure 2**, you'll observe that each candidate rule antecedent-consequent pair has a mirror candidate rule with the antecedent and consequent switched. For example, if an item-set is (2 5 7 9), then one candidate rule with subset size k = 2 is IF (2 7) THEN (5 9) and another candidate rule is IF (5 9) THEN (2 7). So instead of computing antecedents for all possible values of frequent item-set subset size, you can just compute antecedents and consequents for half the possible subset sizes.

Also, method CountInTrans uses a lookup dictionary of saved counts. This is helpful when counting antecedent occurrences because different frequent item-sets can generate the same antecedents. But if frequent item-sets are unique, then the check in the lookup dictionary is a waste of time. Notice the Dictionary parameter is both used and updated so you might want to define it as a ref parameter to make its dual purpose explicit. If you trace through method GetHighConfRules, you'll see several other ways to modify the code.

> Mathematical combinations are fascinating topics in their own right, and somewhat complicated.

One important optimization possibility takes advantage of the fact that, for a given item-set, if a candidate rule doesn't meet the minimum confidence threshold, then any other candidate rule that contains the first rule's consequent can't meet the confidence threshold. For example, suppose a rule IF (2 4 8 9) THEN (3 5) doesn't meet minimum confidence. Then any rule that has (3 5) in its consequent, for example, IF (4 8 9) THEN (2 3 5), will not meet minimum confidence, either.

Wrapping Up

The explanation of association rules and demo code presented in this article should get you up and running if you want to experiment with extracting high-confidence association rules, or create a stand-alone association rule utility program, or add association rule functionality to a software application program.

Unlike several machine learning techniques that are intended to make predictions, association rule learning is an exploratory technique intended to reveal interesting and possibly useful relationships between items. This means you'll have to use a bit of trial and error when finding association rules.

There's a large body of research literature about association rule learning. If you're interested in the theory, I recommend chapter 6 of the book, "Introduction to Data Mining" (Addison-Wesley, 2005) by P. Tan, M. Steinbach, and V. Kumar. Chapter 6 is available for free in PDF format at bit.ly/1m3dbZ9. ■

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 technical expert for reviewing this article: *Richard Hughes and Kirk Olynik*

TED NEWARD

Growl Notification System: Simpler Is Better

In all the histrionics that get stirred up around UI and UX, you often lose sight of the fact that sometimes the best UI isn't the fanciest, most dazzling display of HTML/CSS/JavaScript or mind-blowing animations, but a simple message tucked away in the corner. A lot of what happens, particularly with enterprise systems, are "headless" kinds of operations with no UI.

You still need ways to reach out to the user, though, and ask him to perform actions. You could build a simple little UI utility to give you that, or you could leverage somebody else's utility that's already built and debugged and likely has a lot more features than you'd include.

Growl Basics

Growl (available for download for Windows at growlforwindows.com) is the Windows port of the Mac utility of the same name and is billed as the "Ultimate Notification System." Fundamentally, it isn't difficult to understand. It resides on your machine, hiding out in the notification tray in the lower-right corner of your Windows desktop view, and listens for notifications.

When a message comes through, it pops up a small message box to alert the user. Then you can send these messages across the network as encrypted or password-protected, to avoid having network sniffers watch the traffic. Fundamentally, though, it's about providing the user with notification messages, a la the "toaster" messages you used to see back when instant messaging was hot and MSN Messenger was a thing. That's it, for the most part.

Keep in mind not all developer utilities and solutions have to be these large, grand-scale architectural marvels. In fact, sometimes the most elegant solution to a problem is often a small, singularly focused component that follows the Keep it simple, stupid (KISS) principle. Growl does one thing and does it well: It lets you notify a user (or multiple users, if you start thinking about stretching this across the network) of something happening that would otherwise get past him.

When I first introduced Growl, for example, it was as part of the Oak build system. This was essentially a "continuous build" system. Any time a source file was modified, it would trigger a rebuild of the project. The problem, of course, is if the rebuild isn't something being kicked off by the developer front-and-center staring at Visual Studio, how would the developer know about build problems? The build system sends a notification to Growl, and it surreptitiously displays the build results to the user, tucked away in the corner where it won't demand attention or get in the way of whatever he's doing.

Growling It doesn't take much to think of other situations where this functionality can be useful, both inside and outside a developer-minded context. To developers, this can be useful when long-running tasks (such as builds, data loads, ETL processes and so on) are executing in the background, giving you a heads-up when they're finished. To systems administrators, this can be incredibly useful for critical-error conditions that require near-immediate human intervention but don't warrant terminating operations completely.

For the user, this can be a useful tool for a whole slew of things, including a kind of "push notification" system for applications inside the corporate network, such as when particular data records the user cares about (ones he's currently working on, for example) are being updated, or when events occur within the system (startup, shutdown, system-wide user messages, whatever) need to be pushed in front of the user's eyes. Even networked games could use it, to tell a player when it's his turn.

Growl also lets you "forward" messages to other computers. If a message destined to reach your eyes doesn't find you because you're on lunch, you can configure Growl to send that message to your phone, tablet or group of other computers. You could also have it send an e-mail message or tweet (publicly or direct message) to an account. A user can configure Growl to play sounds (or not) on a per-registered-application basis, set the priority of those notifications on a per-registered-application basis and so on.

Growl may not be much, but it's a pretty useful "not very much" kind of tool. Programming to it is ridiculously simple. Once you've installed Growl, you can send Growl notifications (from batch files or Windows PowerShell scripts, for example) using the command-line "growlnotify" tool installed in the Growl for Windows folder. Kick one off using the following at the command line (assuming C:\Program Files\Growl for Windows, the default Growl installation directory, is on your PATH):

```
growlnotify "This is a test message"
```

Assuming Growl was installed successfully, it will respond at the command line with "Notification sent successfully." A little blue message window will pop up in the corner of your desktop. Open the Growl icon in the system notification tray to examine the configuration options, including the ability to forward messages on to other machines, and use the "/?" command-line parameter to explore how to send Growl notifications across the network. Experiment with some of these options before reading further, because the options seen here pretty much directly reflect the APIs available when writing code to use Growl notifications.

The Growl SDK

The Growl SDK is a thin layer over the Growl Network Transport Protocol (GNTP), a TCP/IP wire protocol strongly reminiscent of HTTP. Given that Growl has been around for a while, it's not surprising there are several libraries built to abstract the details of that protocol. These are collectively referred to as "Growl Connect" libraries.

The same Web site (growlforwindows.com) from which you get the Growl-for-Windows bits also has links to the Microsoft .NET Framework libraries for Growl, as well as links for C++, COM and even SQL Server-to-Growl libraries. (Think about that last one for a moment. It's a package to let you use SQL Server scripts to reach out and send a notification to interested parties, such as DBAs.)

Once the Growl .NET libraries are pulled down, open the .zip file. It contains a sample C# app and a sample Visual Basic .NET app, but what you want specifically are the two .NET assemblies in the "libraries" folder: Growl.CoreLibrary.dll and Growl.Connector.dll. (These are also installed as part of Growl in the Growl home directory, if the SDK download gets lost somewhere.) In any .NET project, simply reference these two as library assemblies, and things are good to go.

> Growl may not be much, but it's a pretty useful "not very much" kind of tool.

Connecting and Registering A Growl client needs to register with Growl before it can send notifications—Growl will ignore any notifications it receives from unregistered applications. Fortunately, not only is this a one-time-only step, but the SDK makes it pretty trivial. Just create a GrowlConnector object (including target hostname and port if reaching out to a remote machine, if desired), and then include some simple information about the application being registered:

```
var connector = new GrowlConnector();
var thisApp = new Application("GrowlCL");
thisApp.Icon = @".\app.png";
```

The icon can be a filename, URL or array of bytes. (Both of these classes, by the way, come from the GrowlConnector namespace.)

Growl thinks of notifications as grouped by type. A Web browser might send "file download started" notifications, "file download completed" notifications and so on. A game might send "new game offered," "player resigned," "your turn," "other player turn completed" and other similar kinds of notifications. Growl wants applications to register these notification types so the user can configure how each kind of notification will be handled. A new Growl client needs to define its notification types and then pass them to the GrowlConnector.Register method to complete registration:

```
// Two kinds of messages: insults and compliments
var insultType = new NotificationType("INSULT", "SICK BURN!");
var compType = new NotificationType("COMPLIMENT", "Nice message");
connector.Register(thisApp, new NotificationType[] { insultType, compType });
```

The first parameter is the string that your code will use to indicate the type of notification. The second is the string that will be displayed in the user's Growl app when viewed. Like the Application object, NotificationType also has an Icon property. This allows for different icons to be displayed along with the text of the message on a per-notification-type basis. It isn't necessary, but certainly helps the final product look more polished.

Note that if the preceding code is run, Growl will pop up a message saying, "GrowlCL has registered" if this is the first time this app has communicated with the Growl app on this machine.

Notify Once the app and its notification types are registered with Growl, it's a pretty simple matter to send a notification. Just create a Notification object, passing in the app name, the notification type string, an optional identifier to uniquely identify this notification (why you might want that I'll get to in just a moment), and the title and body of the message to send, and then send the Notification object off to Growl using the Notify method:

```
var notification = new Notification("GrowlCL", "INSULT", null,
  "YO MAMA!",  "Your mama is so fat, she dropped an apple " +
  "and it entered orbit around her.");
connector.Notify(notification);
```

When the Growl app receives the message, it'll pop up in the lower corner (by default) of the screen.

Yeah, it really is that easy.

Can You Read Me Now? Sometimes you like to know what the user did with the notification. Did she just close it without looking, or did she actually read the message? This can actually be quite important in call centers, where supervisors need to know if employees are in fact reading the daily up-sell special offers, as required.

Growl permits this by registering callbacks, which can either be .NET event handler methods or WebHooks—URLs to which Growl will POST an HTTP request containing data such as the aforementioned ID parameter of the notification sent. In the event the Growl Connector library can't reach the targeted Growl instance, it'll issue an error, but only if the client code has registered an event handler (ErrorResponse) on the GrowlConnector object.

The event handler method will receive an error code and a description of the error that looks almost identical to HTTP error codes and descriptions, making them pretty self-explanatory. The Growl Connector docs (the "Growl for Windows Application Integration Guide" in the downloaded SDK .zip) has a table listing all of these errors, but they're all pretty straightforward ("200 - TIMED OUT," "201 - NETWORK FAILURE" and so on).

Just That Easy

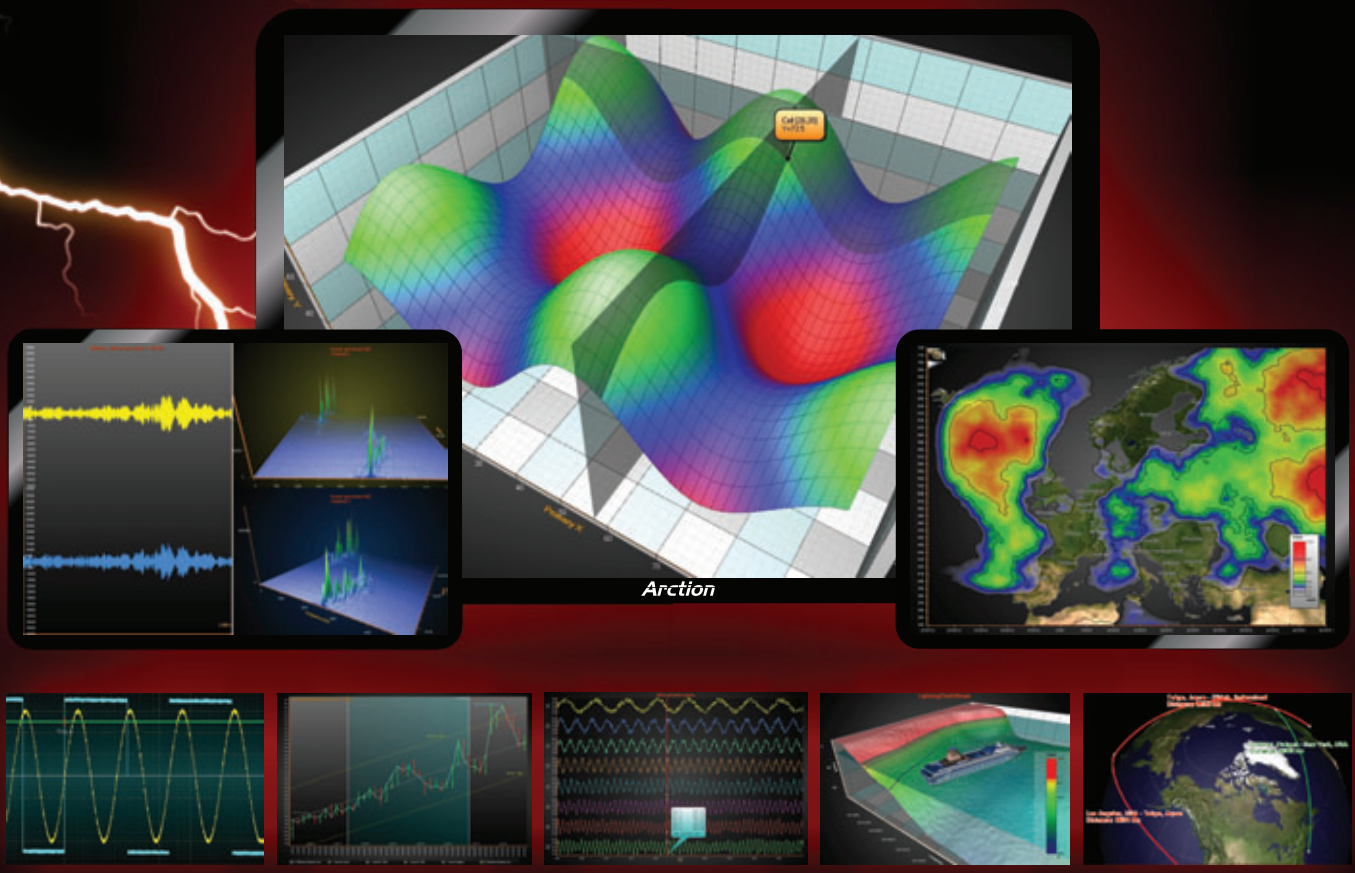
Growl isn't going to win any awards for "most complex architecture." If it can be described almost completely in a single magazine article, then it's definitely a pretty simple, straightforward, low surface-area kind of tool. That, in a nutshell, is probably its greatest strength. Quite frankly, it's the best compliment any technologist can bestow on a library or tool—it's just that easy to use. Happy coding! ■

Ted Neward *is the CTO at iTrellis, a consulting services company. He has written more than 100 articles and authored and coauthored a dozen books, including "Professional F# 2.0" (Wrox, 2010). He's a C# MVP and speaks at conferences around the world. He consults and mentors regularly—reach him at ted@tedneward.com or ted@itrellis.com if you're interested in having him come work with your team, and read his blog at blogs.tedneward.com.*

Thanks to the following technical expert for reviewing this article:
Brian Dunnington (independent consultant)

The fastest rendering data visualization components
for WPF and WinForms...

LightningChart



HEAVY-DUTY DATA VISUALIZATION TOOLS FOR SCIENCE, ENGINEERING AND TRADING

WPF charts performance comparison

Opening large dataset	LightningChart is up to 977,000 % faster
Real-time monitoring	LightningChart is up to 2,700,000 % faster

Winforms charts performance comparison

Opening large dataset	LightningChart is up to 37,000 % faster
Real-time monitoring	LightningChart is up to 2,300,000 % faster

Results compared to average of other chart controls. See details at www.LightningChart.com/benchmark. LightningChart results apply for Ultimate edition.

- Entirely DirectX GPU accelerated
- Superior 2D and 3D rendering performance
- Optimized for real-time data monitoring
- Touch-enabled operations
- Supports gigantic data sets
- On-line and off-line maps
- Great customer support
- Compatible with Visual Studio 2005...2013

Download a free 30-day evaluation from

www.LightningChart.com

Arction

Pioneers of high-performance data visualization

RACHEL APPEL

Design a Cross-Platform Modern App Architecture

Whether you're designing a project from scratch or modifying an existing solution, it's important to know the landscape of the modern app. That means discerning which devices, OSes and browsers are out there that need attention. And "devices" doesn't just mean smartphones, either—that includes tablets, PCs and large desktop screens. Smartphone sales still outnumber the sales of every other type of device, however. Some developers might even have to consider wearable devices as part of their cross-platform architecture.

The bottom line is that writing software these days means writing for a multitude of devices. The primary goal is to share as much code as possible across the various platforms so you have to write the least amount of code. This will help you get the product or app shipped more quickly. It will also be less prone to bugs caused by rewriting the same logic in different languages.

Cross-Platform Device Development

The manner in which you approach cross-platform development will depend on the type of software you already have in place. If you have a Web site from which you want to share content, a hybrid solution might be best. If it's a mature site, you could convert it to a responsive design and support mobile devices. You could forgo the mobile site all together and simply provide a Web site and separate companion native apps. Another factor to consider is whether the app is just another way to present the site's content or if it can stand alone as its own idea.

Build your mobile site first, and then build the native apps. While you develop the native apps, the mobile site can serve as the app until you've covered all the platforms. Those with sites already up and running may need to natively take that content and function to each device. You can continue to grow your mobile user base while developing native apps this way. When you're ready to write your native apps, you might need help deciding which language to use, so be sure to check out my September 2013 column, "Understanding Your Language Choices for Developing Modern Apps" (msdn.microsoft.com/magazine/dn385713).

While it's a popular notion to start developing for the biggest platform or the one with the most device sales, sometimes it's a better business decision to start with the platform that has features running parallel with your requirements. For example, a fitness app that logs food and exercise might have a requirement to show a summary of your meals and activities—myFitnessPal is a good example of this on Windows Phone. This kind of data is perfect for a Live Tile, which is available only on the Windows platform.

You can figure out if your requirements match the platform by listing all your major app features. Review each by verifying it's supported on each platform and possibly to what extent. Where you'll see divergence is with hardware-specific features, such as the camera or OS features such as tiles or voice recognition. Your architectural options from a 30,000-foot view set you up to go in one of three directions: native apps, hybrid apps or HTML5 mobile Web.

Native This means moving full steam ahead on a separate app for each platform. The platforms are likely to be Windows Phone 8, iOS and Android. For some, platforms such as BlackBerry still apply. Each app will take full advantage of the specific features each of the individual platforms provides. Native app performance is great, but, of course, you must remember to write solid and clean code as performance degrades quickly when you're not conscientious. You'll have to write at least the native app UI individually for each platform, so the cost per platform rises to be the most expensive of all these options.

Hybrid Somewhere in that grey area between native and Web is the hybrid app. Hybrid apps usually integrate content from an existing Web site into an app for native packing and deployment. This is especially useful to quickly secure a slot in any of the app stores and have a presence there, perhaps while working on a full native app for that platform. Performance is notoriously slower than with native and Web apps, as hybrid apps are generally constructed by wrapping existing HTML with a container. For example, on the Windows Phone, this is a WebBrowser control. Any time there's an extra layer, there's also potential for performance issues.

The trade-off, however, is speedy deployment on a native platform. Creating hybrid apps requires you use any of the following tools:

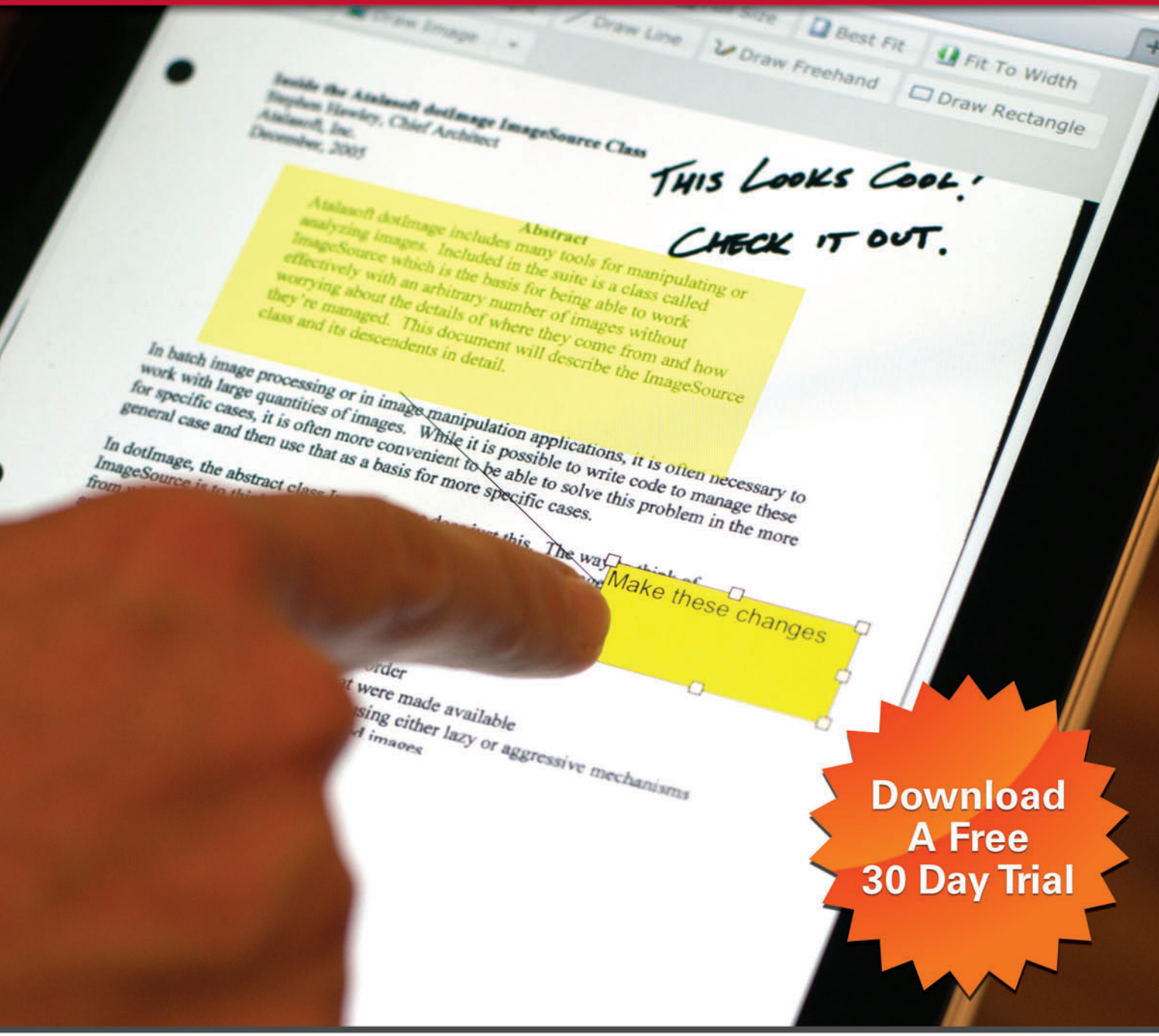
- Apache Cordova/PhoneGap
- Visual Studio Windows Phone HTML5 project template
- Telerik AppBuilder (formerly Icenium)
- Xamarin Studio

Because of their nature, hybrid apps wrap Web content using the WebBrowser control (keeping with the Windows Phone example), which is a XAML element:

```
<phone:WebBrowser x:Name="Browser"
HorizontalAlignment="Stretch"
VerticalAlignment="Stretch"
Loaded="Browser_Loaded"
NavigationFailed="Browser_NavigationFailed" />
```

That's all you need to get started building a hybrid app. On other platforms, the concepts are the same, although the controls might be different. Be cautious while building hybrid apps, as they might not have access or privileges to perform actions that a native app would. This is because of the potential for rogue script from the Web to execute on the client. That's why HTML apps have many of the same restrictions.

Build A Mobile Document Viewer with Annotations, Touch Interfaces, Zooming, Pagination, and More



Download A Free 30 Day Trial

Atalasoft
A Kofax Company

.NET Web Scanning & Imaging SDKs

1.413.572.4443 ◆ www.atalasoft.com

When you build hybrid apps, you can use the Telerik AppBuilder or Apache Cordova in lieu of Visual Studio projects. These third-party tools help close the gaps between the hybrid Web feel and a native look and feel of your app.

HTML5 The benefit of going this route is the immediate wide availability the HTML5 specification brings to the table. Mobile Web sites may have far reach, but lack the native look and feel users want and would normally expect from native apps. Also, an HTML5 app might not be able to access some of the hardware-specific features of its target platforms, such as the Webcam or accelerometer.

HTML5 has the lowest cost per platform, as you should be able to write once and run on most browsers. Some mobile browsers, however, aren't up-to-date with standards support. That can be a major problem when developing HTML5 apps. If native features are important to your project, the lack of access in this area could determine that you go with native apps entirely.

While HTML5 boasts wide availability, keep in mind that in a sense, developing for the Web is similar to developing native apps. You do need to concern yourself with supporting a number of browsers. Like device development, Web developers target those browsers with the most users—Internet Explorer, Chrome, FireFox and Opera.

You want to design rich UXes on the client side on various devices, and you can't reuse that specific device code. The following native features are unlikely to behave consistently or be available across platforms:

- Push Notifications
- Wallet
- Near Field Communication (NFC)
- Multitasking/multithreading
- Speech
- Navigation Schemes
- Dialogs

In hybrid or HTML5 client apps, don't rely on access to these native-specific attributes. If features like these are part of your app requirements, you'll need to go the all-native route.

Architect Cross-Platform Solutions

Determining which platforms you'll support is just one decision in the app landscape. You must also design the solution, and take into consideration things such as time to market and for which platform to develop first. How to share as much code as possible between apps is also an important consideration.

The back-end Web services and APIs work well in a cross-platform scenario because every platform supports HTTP. Using layers of services in the back end means you only need to write the code for those services once.

As far as expanding the architecture to focus on the client, you'll want to share as much code as possible. You can easily code on Windows 8 and Windows Phone with Portable Class Libraries (PCLs), but that leaves out other platforms such as Android and iOS. You can share code across Windows 8, Windows Phone, Android, and iOS by writing in C# and using Xamarin tools to cross-compile.

The point of going native is so users get the same look and feel of the native platform. While tools such as Xamarin will let you share some code in the UI layer, you'll get the best UX results if you handcraft each UI separately. That usually means modifying the generated Xamarin code. This is a good time to add any platform-specific features, so users on each platform get a customized experience.

The App Code Layer maps to the controller or ViewModel parts of the Model-View-Controller (MVC) or Model-View-ViewModel (MVVM) patterns. The UI layer maps to the View in both MVC and MVVM. The Model, of course, is the data representation. This might be JavaScript classes that map to database objects in Local Storage, or it might map to remote schema back in the database. Local data isn't something you'd normally share between devices, as it often includes device-specific information.

The path of least resistance in building your solution is to start with the back-end layers first, then create a Web site, then the apps. This way you get something out the door to the widest range possible. On the apps side, you'll need to determine which to build first and if any should be your primary platform.

> Determining which platforms you'll support is just one decision in the app landscape.

I'm fond of the Windows 8/Windows RT/Windows Phone platform myself and think it (specifically Windows Phone) has the best UX of all the devices available. That's not just because I've worked for Microsoft. I think the iPhone has a good UX, too. Features such as Tiles and top-of-the-line Nokia cameras are convincing reasons to make Windows Phone your feature platform for anyone who needs that kind of functionality in an app. Also, Windows Phone seamlessly manages and merges your contacts, calendar and other vital data across Windows Phone-based devices with the built-in cloud-based people and calendar apps.

Wrapping Up

With the advent of the bring your own device (BYOD) movement in workplaces everywhere, native app development has become commonplace. As you can see, there are many variables and things to consider when building apps that can affect your success. Unless you're working for a corporation that requires a standard desktop client, then I'd advise against it. This amounts to not being the first app on that platform.

If you have a generous budget and personnel, plus an extended time frame to build your solution, you can create a Web site with a mobile version and a complete set of native apps. That means you're providing every possible way to access your services and data. Of course, this route is the most work and the most expensive. ■

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 technical experts for reviewing this article:
Frank La Vigne (Microsoft) and Wally McClure (Scalable Development)

That's right, we're transporting Visual Studio Live! to our nation's capital for the first time in 21 years. From Oct 6 – 9, 2014, developers, software architects, engineers and designers will gather for 4 days of cutting-edge education on the Microsoft Platform.

The Washington Marriott Metro Center, our headquarters, is centrally located in the heart of it all: the Mall and National Archives to the South; the White House the West; Mt Vernon Square to the North; and Metro Center to the East. You can explore this country's rich history just steps from the front door – all while sharpening your .NET skills and your ability to build better apps!

SUPPORTED BY

Microsoft | Visual Studio | msdn magazine | Visual Studio MAGAZINE

PRODUCED BY

1105 MEDIA

Tracks Include:

- Visual Studio/.NET Framework
- Windows Client
- JavaScript/HTML5 Client
- ASP.NET
- Cloud Computing
- Windows Phone
- Cross-Platform Mobile Development
- SharePoint/Office
- SQL Server

YOUR GUIDE TO THE .NET DEVELOPMENT UNIVERSE

REGISTER TODAY AND SAVE $300!

Use promo code VSLDC2 by August 13

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

CONNECT WITH VISUAL STUDIO LIVE!

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

linkedin.com – Join the "Visual Studio Live" group!

vslive.com/dc

CHARLES PETZOLD

Manipulating Triangles in 3D Space

The Dutch graphic artist M. C. Escher has always been a particular favorite among programmers, engineers and other techies. His witty drawings of impossible structures play with the mind's need to impose order on visual information, while his use of mathematically inspired meshed patterns seems to suggest a familiarity with software recursion techniques.

I'm a particular fan of Escher's mix of two-dimensional and three-dimensional imagery, such as "Drawing Hands" from 1948, in which a pair of 3D hands arise from a 2D drawing that is itself being sketched by the 3D hands (see **Figure 1**). But this juxtaposition of the 2D and 3D images emphasizes how the 3D hands only appear to have depth as a result of detail and shading. Obviously, everything in the drawing is rendered on flat paper.

I want to do something similar in this article: I want to make 2D graphical objects seem to acquire depth and body as they arise from the screen and float in 3D space, and then retreat back into the flat screen.

These graphical objects won't be portrayals of human hands, however. Instead, I'll stick with perhaps the simplest 3D objects—the five Platonic solids. The Platonic solids are the only possible convex polyhedra whose faces are identical regular convex polygons, with the same number of faces meeting at each vertex. They are the tetrahedron (four triangles), octahedron (eight triangles), icosahedron (20 triangles), cube (six squares) and dodecahedron (12 pentagons).

Platonic solids are popular in rudimentary 3D graphics programming because they're generally easy to define and assemble. Formulas for the vertices can be found in Wikipedia, for example.

To make this exercise as pedagogically gentle as possible, I'll be using Direct2D rather than Direct3D. However, you'll need to become familiar with some concepts, data types, functions and structures often used in connection with Direct3D.

My strategy is to define these solid objects using triangles in 3D space, and then apply 3D transforms to rotate them. The transformed triangle coordinates are then flattened into 2D space by ignoring the Z coordinate, where they're used to create ID2D1Mesh objects, which are then rendered using the FillMesh method of the ID2D1DeviceContext object.

Figure 1 **M.C. Escher's "Drawing Hands"**

As you'll see, it's not enough to simply define coordinates for 3D objects. Only when shading is applied to mimic the reflection of light do the objects seem to escape the flatness of the screen.

Code download available at msdn.microsoft.com/magazine/msdnmag0514.

3D Points and Transforms

This exercise requires that 3D matrix transforms be applied to 3D points to rotate objects in space. What are the best data types and functions for this job?

Interestingly, Direct2D has a D2D1_MATRIX_4X4_F structure and a Matrix4x4F class in the D2D1 namespace that are suitable for representing 3D transform matrices. However, these data types are designed only for use with the DrawBitmap methods defined by ID2D1DeviceContext, as I demonstrated in the April installment of this column. In particular, Matrix4x4F doesn't even have a method named Transform that can apply the transform to a 3D point. You'd need to implement that matrix multiplication with your own code.

A better place to look for 3D data types is the DirectX Math library, which is used by Direct3D programs, as well. This library defines more than 500 functions—all of which begin with the letters XM—and several data types. These are all declared in the DirectXMath.h header file and associated with a namespace of DirectX.

Every single function in the DirectX Math library involves the use of a data type named XMVECTOR, which is a collection of four numbers. XMVECTOR is suitable for representing 2D or 3D points (with or without a W coordinate) or a color (with or without an alpha channel). Here's how you'd define an object of type XMVECTOR:

```
XMVECTOR vector;
```

Figure 2 **The Data Structures Used for Storing 3D Figures**

```
struct RenderInfo
{
  Microsoft::WRL::ComPtr<ID2D1Mesh> mesh;
  Microsoft::WRL::ComPtr<ID2D1SolidColorBrush> brush;
};

struct FigureInfo
{
  // Constructor
  FigureInfo()
  {
  }

  // Move constructor
  FigureInfo(FigureInfo && other) :
    srcTriangles(std::move(other.srcTriangles)),
    dstTriangles(std::move(other.dstTriangles)),
    renderInfo(std::move(other.renderInfo))
  {
  }

  int faceCount;
  int trianglesPerFace;
  std::vector<Triangle3D> srcTriangles;
  std::vector<Triangle3D> dstTriangles;
  D2D1_COLOR_F color;
  std::vector<RenderInfo> renderInfo;
};

std::vector<FigureInfo> m_figureInfos;
```

Migrate and Sync HPQC to TFS and MTM with The Kovair Omnibus Platform

Why Migrate

- **O**pt for New Tool Strategies
- **M**anage Development and QA Teams Better
- **N**eed for Maintenance Cost Reduction
- **I**ncrease Efficiency in Development Process
- **B**uild Single Repository for All ALM Assets
- **U**nify with the Rest of Microsoft Tools Family
- **S**ave with Productivity Gains from Integrated Tools

How Kovair Helps Migrate with Omnibus

- Complete support for "Call 2 Tests"
- HPQC Requirements Folder Structure to TFS Area Path for Requirement tracking
- HPQC Releases & Cycles to TFS Iterations
- Migration of HPQC Design Steps with Parameters and Default Value
- Complete support of Manual and Automated Testing from MTM
- Codeless Configuration of Integration rules

For More Details Visit: http://goo.gl/dwLeLj

KOVAIR

www.kovair.com
email: sales@kovair.com

+1.408.262.0200 Press 1 [USA]
+91-33-4065 7016 / 17 [India]

Figure 3 **Defining the Tetrahedron**

```
FigureInfo tetrahedron;
tetrahedron.faceCount = 4;
tetrahedron.trianglesPerFace = 1;
tetrahedron.srcTriangles =
{
  Triangle3D { XMFLOAT4(-1,  1, -1, 1),
               XMFLOAT4(-1, -1,  1, 1),
               XMFLOAT4( 1,  1,  1, 1) },

  Triangle3D { XMFLOAT4( 1, -1, -1, 1),
               XMFLOAT4( 1,  1,  1, 1),
               XMFLOAT4(-1, -1,  1, 1) },

  Triangle3D { XMFLOAT4( 1,  1,  1, 1),
               XMFLOAT4( 1, -1, -1, 1),
               XMFLOAT4(-1,  1, -1, 1) },

  Triangle3D { XMFLOAT4(-1, -1,  1, 1),
               XMFLOAT4(-1,  1, -1, 1),
               XMFLOAT4( 1, -1, -1, 1) }
};

tetrahedron.srcTriangles.shrink_to_fit();
tetrahedron.dstTriangles.resize(tetrahedron.srcTriangles.size());
tetrahedron.color = ColorF(ColorF::Magenta);
tetrahedron.renderInfo.resize(tetrahedron.faceCount);
m_figureInfos.at(0) = tetrahedron;
```

Notice I said XMVECTOR is a collection of "four numbers" rather than "four floating-point values" or "four integers." I can't be more specific because the actual format of the four numbers in an XMVECTOR object is hardware-dependent.

XMVECTOR is not a normal data type! It's actually a proxy for four hardware registers on the processor chip, specifically single instruction multiple data (SIMD) registers used with streaming SIMD extensions (SSE) that implement parallel processing. On x86 hardware these registers are indeed single-precision floating-point values, but in ARM processors (found in Windows RT devices) they're integers defined to have fractional components.

For this reason, you shouldn't attempt to access the fields of an XMVECTOR object directly (unless you know what you're doing). Instead, the DirectX Math library includes numerous functions to set the fields from integer or floating point values. Here's a common one:

```
XMVECTOR vector = XMVectorSet(x, y, z, w);
```

Functions also exist to obtain the individual field values:

```
float x = XMVectorGetX(vector);
```

Because this data type is a proxy for hardware registers, certain restrictions govern its use. Read the online "DirectXMath Programming Guide" (bit.ly/1d4L7Gk) for details on defining structure members of type XMVECTOR and passing XMVECTOR arguments to functions.

In general, however, you'll probably use XMVECTOR mostly in code that's local to a method. For the general-purpose storage of 3D points and vectors, the DirectX Math library defines other data types that are simple normal structures, such as XMFLOAT3 (which has three data members of type float named x, y and z) and XMFLOAT4 (which has four data members to include w). In particular, you'll want to use XMFLOAT3 or XMFLOAT4 for storing arrays of points.

Figure 4 **The PlatonicSolids Program As It Begins Running**

It's easy to transfer between XMVECTOR and XMFLOAT3 or XMFLOAT4. Suppose you use XMFLOAT3 to store a 3D point:

```
XMFLOAT3 point;
```

When you need to use one of the DirectX Math functions that require an XMVECTOR, you can load the value into an XMVECTOR using the XMLoadFloat3 function:

```
XMVECTOR vector = XMLoadFloat3(&point);
```

The w value in the XMVECTOR is initialized to 0. You can then use the XMVECTOR object in various DirectX Math functions. To store the XMVECTOR value back in the XMFLOAT3 object, call:

```
XMStoreFloat3(&point, vector);
```

Similarly, XMLoadFloat4 and XMStoreFloat4 transfer values between XMVECTOR objects and XMFLOAT4 objects, and these are often preferred if the W coordinate is important.

In the general case, you'll be working with several XMVECTOR objects in the same block of code, some of which correspond to underlying XMFLOAT3 or XMFLOAT4 objects, and some of which are just transient. You'll see examples shortly.

I said earlier that every function in the DirectX Math library involves XMVECTOR. If you've explored the library, you might find some functions that actually don't require an XMVECTOR but do involve an object of type XMMATRIX.

The XMMATRIX data type is a 4×4 matrix suitable for 3D transforms, but it's actually four XMVECTOR objects, one for each row:

```
struct XMMATRIX
{
  XMVECTOR r[4];
};
```

So what I said was correct because all the DirectX Math functions that require XMMATRIX objects really do involve XMVECTOR objects, as well, and XMMATRIX has the same restrictions as XMVECTOR.

Just as XMFLOAT4 is a normal structure you can use to transfer values to and from an XMVECTOR object, you can use a normal structure named XMFLOAT4X4 to store a 4×4 matrix, and transfer that to and from an XMMATRIX using the XMLoadFloat4x4 and XMStoreFloat4x4 functions.

If you've loaded a 3D point into an XMVECTOR object (named vector, for example), and you've loaded a transform matrix into an XMMATRIX object named matrix, you can apply that transform to the point using:

```
XMVECTOR result = XMVector3Transform(vector, matrix);
```

Or, you can use:

```
XMVECTOR result = XMVector4Transform(vector, matrix);
```

The only difference is that XMVector4Transform uses the actual w value of the XMVECTOR while XMVector3Transform assumes that it's 1, which is correct for implementing 3D translation.

However, if you have an array of XMFLOAT3 or XMFLOAT4 values and you want to apply the transform to the entire array, there's a much better solution: The XMVector3TransformStream and XMVector4TransformStream functions apply the XMMATRIX to an array of values and store the results in an array of XMFLOAT4 values (regardless of the input type).

Figure 5 **Rotating the Figures**

```
// Calculate total matrix
XMMATRIX matrix = XMMatrixScaling(scale, scale, scale) *
                  XMMatrixRotationX(xAngle) *
                  XMMatrixRotationY(yAngle) *
                  XMMatrixRotationZ(zAngle);

// Transform source triangles to destination triangles
for (FigureInfo& figureInfo : m_figureInfos)
{
  XMVector4TransformStream(
    (XMFLOAT4 *) figureInfo.dstTriangles.data(),
    sizeof(XMFLOAT4),
    (XMFLOAT4 *) figureInfo.srcTriangles.data(),
    sizeof(XMFLOAT4),
    3 * figureInfo.srcTriangles.size(),
    matrix);
}
```

The bonus: Because XMMATRIX is actually in the SIMD registers on a CPU that implements SSE, the CPU can use parallel processing to apply that transform to the array of points, and accelerate one of the biggest bottlenecks in 3D rendering.

Defining Platonic Solids

The downloadable code for this column is a single Windows 8.1 project named PlatonicSolids. The program uses Direct2D to render 3D images of the five Platonic solids.

Like all 3D figures, these solids can be described as a collection of triangles in 3D space. I knew I'd want to use XMVector3TransformStream or XMVector4TransformStream to transform an array of 3D triangles, and I knew the output array of these two functions is always an array of XMFLOAT4 objects, so I decided to use XMFLOAT4 for the input array, as well, and that's how I defined my 3D triangle structure:

```
struct Triangle3D
{
  DirectX::XMFLOAT4 point1;
  DirectX::XMFLOAT4 point2;
  DirectX::XMFLOAT4 point3;
};
```

Figure 2 shows some additional private data structures defined in PlatonicSolidsRenderer.h that store information necessary to describe and render a 3D figure. Each of the five Platonic solids is an object of type FigureInfo. The srcTriangles and dstTriangles collections store the original "source" triangles and the "destination" triangles after scaling and rotation transforms have been applied. Both collections have a size equal to the product of faceCount and trianglesPerFace. Notice that srcTriangles.data and dstTriangles.data are effectively pointers to XMFLOAT4 structures and can therefore be arguments to the XMVector4TransformStream function. As you'll see, this happens during the Update method in the PlatonicSolidRenderer class.

The renderInfo field is a collection of RenderInfo objects, one for each face of the figure. The two members of this structure are also determined during the Update method, and they're simply passed to the FillMesh method of the ID2D1DeviceContext object during the Render method.

The constructor of the PlatonicSolidsRenderer class initializes each of the five FigureInfo objects. **Figure 3** shows the process for the simplest of the five, the tetrahedron.

The initialization of the octahedron and icosahedron are similar. In all three cases, each face consists of just one triangle. In terms of pixels, the coordinates are very small, but code later in the program scales them to a proper size.

The cube and dodecahedron are different, however. The cube has six faces, each of which is a square, and the dodecahedron is 12 pentagons. For these two figures, I used a different data structure to store the vertices of each face, and a common method that converted each face into triangles—two triangles for each face of the cube and three triangles for each face of the dodecahedron.

For ease in converting the 3D coordinates into 2D coordinates, I've based these figures on a coordinate system in which positive X coordinates increase to the right and positive Y coordinates increase going down. (It's more common in 3D programming for positive Y coordinates to increase going up.) I've also assumed that positive Z coordinates come out of the screen. Therefore, this is a left-hand coordinate system. If you point the forefinger of your left hand in the direction of positive X, and the middle finger in the direction of positive Y, your thumb points to positive Z.

The viewer of the computer screen is assumed to be located at a point on the positive Z axis looking toward the origin.

Rotations in 3D

The Update method in PlatonicSolidsRenderer performs an animation that consists of several sections. When the program begins running, the five Platonic solids are displayed, but they appear to be flat, as shown in **Figure 4**.

These are obviously not recognizable as 3D objects!

In 2.5 seconds, the objects begin rotating. The Update method calculates rotation angles and a scaling factor based on the size of the screen, and then makes use of DirectX Math functions. Functions such as XMMatrixRotationX compute an XMMATRIX object representing rotation around the X axis. XMMATRIX also defines matrix multiplication operators so the results of these functions can be multiplied together.

Figure 5 shows how a total matrix transform is calculated and applied to the array of Triangle3D objects in each figure.

Once the figures begin rotating, however, they still appear to be flat polygons, even though they're changing shape.

Occlusion and Hidden Surfaces

One of the crucial aspects of 3D graphics programming is making sure objects closer to the viewer's eye obscure (or occlude) objects further away. In complex scenes, this isn't a trivial problem, and, generally, this must be performed in graphics hardware on a pixel-by-pixel basis.

With convex polyhedra, however, it's relatively quite simple. Consider a cube. As the cube is rotating in space, mostly you see three faces, and sometimes just one or two. Never do you see four, five or all six faces.

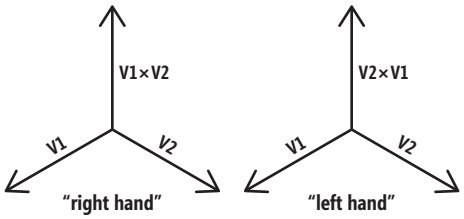


Figure 6 **The Vector Cross Product**

For a particular face of the rotating cube, how can you determine what faces you see and what faces are hidden? Think about vectors (often visualized as arrows with a particular direction) perpendicular to each face of the cube, and pointing to the outside of the cube. These are referred to as "surface normal" vectors.

Only if a surface normal vector has a positive Z component will that surface be visible to a viewer observing the object from the positive Z axis.

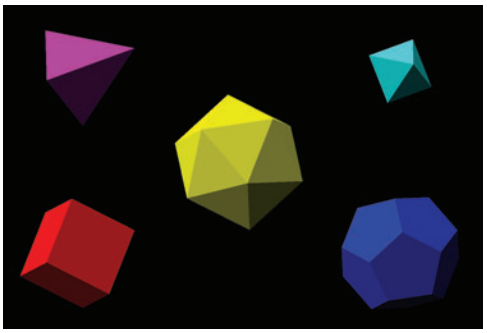


Figure 7 **The PlatonicSolids Program with Maximum 3D**

Mathematically, computing a surface normal for a triangle is straightforward: The three vertices of the triangle define two vectors, and two vectors (V1 and V2) in 3D space define a plane, and a perpendicular to that plane is obtained from the vector cross product, as shown in **Figure 6**.

The actual direction of this vector depends on the handedness of the coordinate system. For a right-hand coordinate system, for example, you can determine the direction of the V1×V2 cross product by curving the fingers of your right hand from V1 to V2. The thumb points in the direction of the cross product. For a left-hand coordinate system, use your left hand.

For any particular triangle that makes up these figures, the first step is to load the three vertices into XMVECTOR objects:

```
XMVECTOR point1 = XMLoadFloat4(&triangle3D.point1);
XMVECTOR point2 = XMLoadFloat4(&triangle3D.point2);
XMVECTOR point3 = XMLoadFloat4(&triangle3D.point3);
```

Then, two vectors representing two sides of the triangle can be calculated by subtracting point2 and point3 from point1 using convenient DirectX Math functions:

```
XMVECTOR v1 = XMVectorSubtract(point2, point1);
XMVECTOR v2 = XMVectorSubtract(point3, point1);
```

All the Platonic solids in this program are defined with triangles whose three points are arranged clockwise from point1 to point2 to point3 when the triangle is viewed from outside the figure. A surface normal pointing to outside the figure can be calculated using a DirectX Math function that obtains the cross product:

```
XMVECTOR normal = XMVector3Cross(v1, v2);
```

A program displaying these figures could simply choose to not display any triangle with a surface normal that has a 0 or negative Z component. The PlatonicSolids program instead continues to display those triangles but with a transparent color.

It's All About the Shading

You see objects in the real world because they reflect light. Without light, nothing is visible. In many real-world environments, light comes from many different directions because it bounces off other surfaces and diffuses in the air.

In 3D graphics programming, this is known as "ambient" light, and it's not quite adequate. If a cube is floating in 3D space and the same ambient light strikes all six faces, all six faces would be colored the same and it wouldn't look like a 3D cube at all.

Scenes in 3D, therefore, usually require some directional light—light coming from one or more directions. One common approach for simple 3D scenes is to define a directional light source as a vector that seems to come from behind the viewer's left shoulder:

```
XMVECTOR lightVector = XMVectorSet(2, 3, -1, 0);
```

From the viewer's perspective, this is one of many vectors that points to the right and down, and away from the viewer in the direction of the negative Z axis.

In preparation for the next job, I want to normalize both the surface normal vector and the light vector:

```
normal = XMVector3Normalize(normal);
lightVector = XMVector3Normalize(lightVector);
```

The XMVector3Normalize function calculates the magnitude of the vector using the 3D form of the Pythagorean Theorem, and then divides the three coordinates by that magnitude. The resultant vector has a magnitude of 1.

If the normal vector happens to be equal to the negative of the lightVector, that means the light is striking the triangle perpendicular to its surface, and that's the maximum illumination that directional light can provide. If the directional light isn't quite perpendicular to the triangle surface, the illumination will be less.

Mathematically, the illumination of a surface from a directional light source is equal to the cosine of the angle between the light vector and the negative surface normal. If these two vectors have a magnitude of 1, then that crucial number is provided by the dot product of the two vectors:

```
XMVECTOR dot = XMVector3Dot(normal, -lightVector);
```

The dot product is a scalar—one number—rather than a vector, so all the fields of the XMVECTOR object returned from this function hold the same values.

To make it seem as if the rotating Platonic solids magically assume 3D depth as they arise from the flat screen, the PlatonicSolids program animates a value called lightIntensity from 0 to 1 and then back to 0. The 0 value is no directional light shading and no 3D effect, while the 1 value is maximum 3D. This lightIntensity value is used in conjunction with the dot product to calculate a total light factor:

```
float totalLight = 0.5f +
  lightIntensity * 0.5f * XMVectorGetX(dot);
```

The first 0.5 in this formula refers to ambient light, and the second 0.5 allows totalLight to range from 0 to 1 depending on the value of the dot product. (Theoretically, this isn't quite correct. Negative values of the dot product should be set to 0 because they result in total light that is less than ambient light.)

This totalLight is then used to calculate a color and brush for each face:

```
renderColor = ColorF(totalLight * baseColor.r,
                     totalLight * baseColor.g,
                     totalLight * baseColor.b);
```

The result with maximum 3D-ishness is shown in **Figure 7**. ■

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*

Mayday!

Santa brought me an Amazon Kindle Fire HDX tablet this past Christmas. I especially like its groundbreaking feature called the Mayday button. In case you've never seen a war or disaster movie in your life, "Mayday" is the international radio distress call, from the French "M'aidez," meaning "help me." When you can't make your Kindle do what you want, you just tap that button and a live human being appears on your screen to help you.

I tried it on Christmas day. It took eight or nine seconds to connect me to a guy also named Dave, whom I could see in a small window. He could see my screen contents but he couldn't see me through the Kindle's camera (he said). He solved my problem (music from songza.com wouldn't play, but local MP3s would) rather quickly, drawing on my Kindle screen saying, "Tap here, then tap there, then select this thing and that should do it." Indeed it did.

> I wonder if customers will start using the Mayday button to unburden their souls, as they might to a priest, a shrink or a good bartender.

I tried the Mayday button a few more times on subsequent days. Connecting never took more than 10 seconds. Support rep Elaine quickly fixed my browser from freezing at lowes.com, but couldn't help me get past Level 28 on Candy Crush. She did offer some general hints, though: "Try to make matches as low down in the stack as you can, so the new candies dropping in from the top make more matches as they fall."

This is revolutionary. You get: a live, English-speaking human being; automatically connected to your specific device; more or less instantly at the tap of a button; at no extra cost, as part of the basic functionality of your consumer-grade product. It's a quantum leap better than what I get on my Nexus, my Surface or even my iPad.

This feature can't be cheap to provide. Amazon probably figures the Mayday button will convince customers to buy Kindles instead of other tablets, which in turn will lead them to buy more content from Amazon, probably driving the final stake into the heart of Barnes & Noble, thereby sending even more customers to Amazon. It's sort of like selling word processors along with PC OSes, which I think I remember hearing about once or twice.

Having a live human look at your Amazon history might worry some users. I suspect readers felt more comfortable ordering "The Sex-Starved Marriage" by Michelle Weiner Davis remotely from Amazon, rather than plunking it down in front of a human cashier at a bookstore. But even though I'm talking to another human, Kindle still feels anonymous to me. Like a parishioner talking to a priest in a confessional, I can see him, but he can't see me.

I wonder if customers will start using the Mayday button to unburden their souls, as they might to a priest, a shrink or a good bartender. Your regular bartender remembers your drink order and your troubles from one session to the next; that's why you frequent that bar. Amazon could scale this up very easily. Instead of storing state (your preferences and troubles) in specific object instances (Charlie the bartender), Amazon keeps it in a central database. When you cry "Mayday!" the next available customer rep (stateless object instance) retrieves your state from the central store, as though the substitute bartender Stacy could pull up your drinking record on Charlie's day off.

"Yes sir, here's your regular: a Churchill martini—chilled gin with a glance at an unopened vermouth bottle. And I see here that your wife doesn't understand you, eh? Mine neither. OK, I'll make that a double." It's a much more scalable architecture.

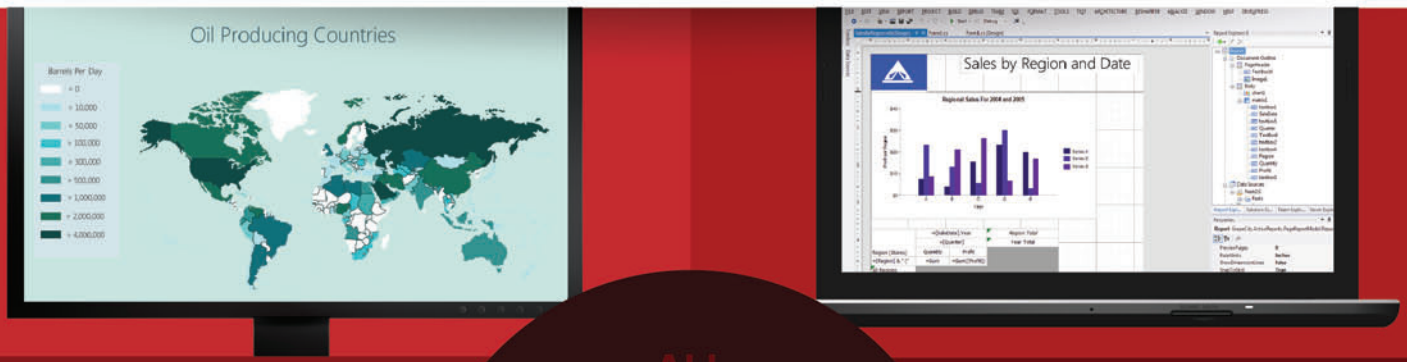
You tip your bartender for knowing you well and putting up with you. I can see Amazon extending that. "The tech support is free, but it's five bucks more if I have to listen to your sob story. I can charge it right to your Amazon account."

If anyone in the world can recognize a novel idea for profit, it's Amazon founder Jeff Bezos. I think Siri had better watch out. ■

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.*

REPORTS ANYWHERE

We are the Reporting experts and we care about performance and Visual Studio integration as much as you. The all new ActiveReports 8 reporting engine delivers a granular API that allows you to create concise and beautiful reports. Whether you need multiple outputs, the ability to create maps, barcodes, or the power of HTML5 to display your reports, ActiveReports 8 can handle all of your heavy reporting requirements.



Sophisticated, Fast & Powerful Reports

ALL 8 NEW **ActiveReports**

Flexible Layouts Using Section, Region & Fixed Page Designers

HTML5 & Touch Optimized Report Viewers for Mobile Devices

A New Scalable, Distributed & Load Balanced Enterprise-grade Report Server

ComponentOne®
a division of GrapeCity®

DOWNLOAD YOUR FREE TRIAL
▸ componentone.com

Aspose.Cells

Work with spreadsheets and data without depending on Microsoft Excel

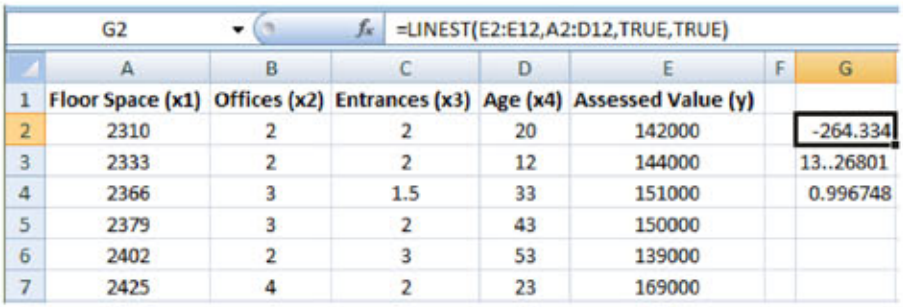
- Solution for spreadsheet creation, manipulation and conversion.
- Import and export data.

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.

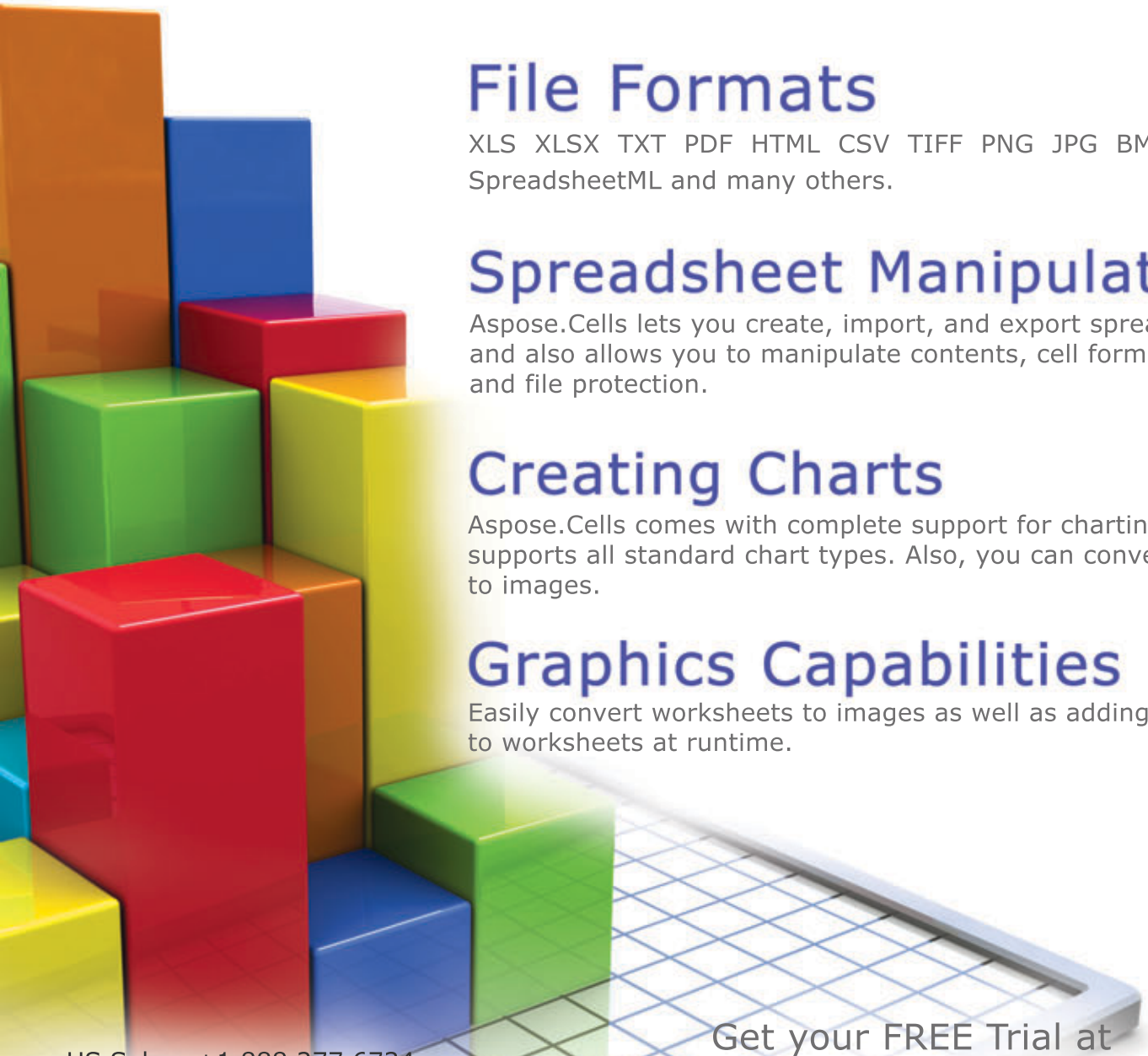
www.aspose.com

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

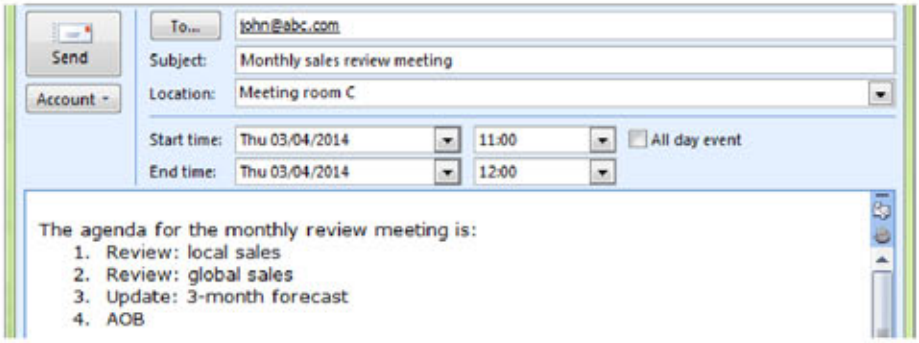
- Complete email processing solution.
- Message file format support.

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

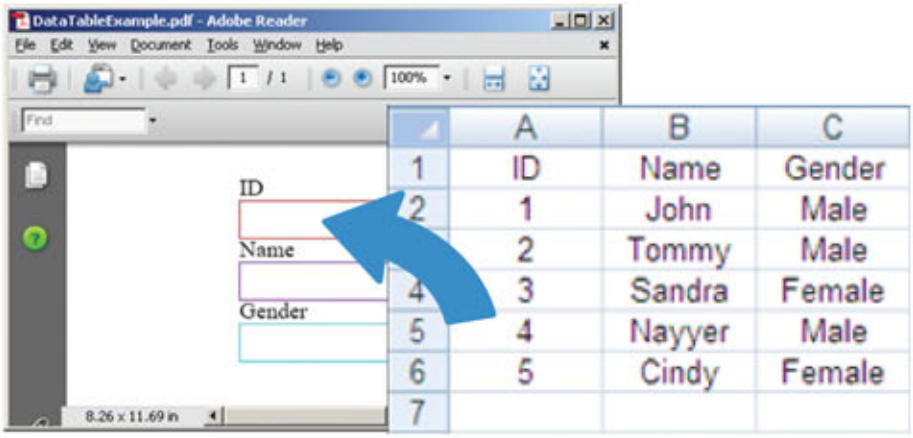
- A complete solution for programming with PDF files.
- Work with PDF forms and form fields.

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