THE MICROSOFT JOURNAL FOR DEVELOPERS

OCTOBER 2013 VOL 28 NO 10

msdn magazine

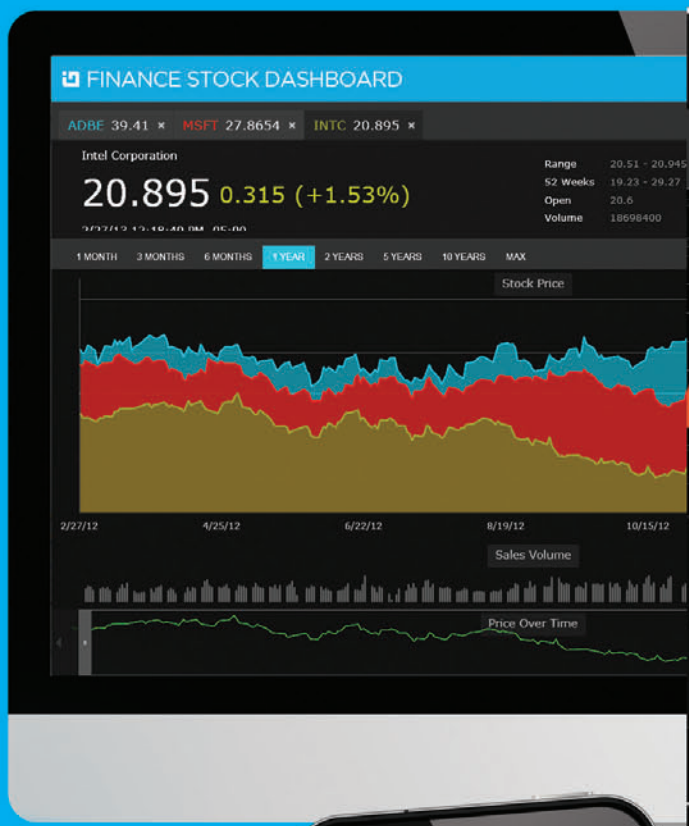
Build Modern Web Applications with Katana...30

COLUMNS

Microsoft

Desktop

Deliver high performance, scalable and stylable touch-enabled enterprise applications in the platform of your choice.



Native Mobile

Develop rich, device-specific user experience for iOS, Android, and Windows Phone, as well as mobile cross-platform apps with Mono-Touch.

Download Your Free Trial

infragistics.com/enterprise-READY

Dr.Dobb's JOLT 2013 BEST PROGRAMMER LIBRARIES

Cross-Device

Build standards-based, touch-enabled HTML5 & jQuery experiences for desktop, tablet, and mobile delivery, including multi-device targeting with frameworks such as PhoneGap and MVC.

UX Prototyping

Explore design ideas through rapid, user-centered wireframing, prototyping, and evaluation before investing a single line of code.

INFRAGISTICS
DESIGN / DEVELOP / EXPERIENCE

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

msdn magazine

OCTOBER 2013 VOLUME 28 NUMBER 10

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

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

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

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

List Rental: This publication's subscriber list, as well as other lists from 1105 Media, Inc., is available for rental. For more information, please contact our list manager, Jane Long, Merit Direct. Phone: 913-685-1301; E-mail: jlong@meritdirect.com; Web: www.meritdirect.com/1105

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

Microsoft

BPA WORLDWIDE

Printed in the USA

LEADTOOLS®

The world's leading Imaging SDK
RUNS ANYWHERE

Document	Medical	Multimedia	Imaging
OCR, Barcode & Forms Recognition	*DICOM*	*Playback, Capture & Conversion*	*Viewers*
PDF Read, Write & Edit	*PACS*	*MPEG-2 Transport Stream*	*Image processing*
Cleanup and Preprocessing	*Medical Workstation*	*DVR*	*150+ Formats*

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

EDITOR'S NOTE

MICHAEL DESMOND

Microsoft Swings Its Web Sword

This month's issue of *MSDN Magazine* leads off with Howard Dierking's exploration of the Microsoft Katana Project ("Getting Started with the Katana Project"), an intriguing effort to turn the old framework-based model of application development on its ear. Based on the open source Open Web Interface for .NET (OWIN) project that defines an interface to decouple Web applications from Web servers, Katana presents a set of OWIN components—including infrastructure and functional components—that are provided by Microsoft.

As Dierking, a program manager on the Windows Azure Frameworks and Tools team, explains in his article this month, Katana enables developers to orchestrate different frameworks together into a single Web application. Katana defines interactions among Web servers and application components, Dierking writes, by reducing "all interactions to a small set of types and a single function signature known as the application delegate, or AppFunc."

It's an important effort in the arena of cloud-enabled Web application development, where monolithic frameworks such as the Microsoft .NET Framework simply cannot evolve quickly enough to address emergent challenges. The release of ASP.NET MVC and ASP.NET Web API by Microsoft helped decouple key Web application development frameworks from the long cycles of the .NET Framework. Katana takes things an important step further, by allowing developers to mix and match application components from different frameworks.

"The thing that's exciting about Katana is that it goes beyond frameworks and libraries and enables a tremendous amount of agility for releasing core infrastructure components like servers and hosts," Dierking explained in an interview, noting that the Katana OWIN pipeline model gives developers control over what constitutes the framework.

"Practically, this means that developers don't end up paying a performance cost for framework features that aren't in use. It also means that gone are the days of waiting for a new framework release in order to take advantage of a new capability," Dierking said. "With Katana, simply install the new capability via NuGet and add it to the OWIN pipeline."

Dierking said developers are sometimes confused about the role of Katana in the Web application stack. Applications, he said, don't need to be rewritten, because Katana isn't an application development framework. "It's a way to compose one or more application development frameworks together. If you write ASP.NET Web APIs today, you'll continue to do so when they're hosted with Katana host and server components," Dierking said.

Dierking said that developers he talks to are most excited about being able to compose different frameworks together into a single Web application and to take a Web application and easily switch servers and hosts. The componentized stack, he said, enables a "rich component ecosystem that can innovate faster than any traditional framework could ever hope."

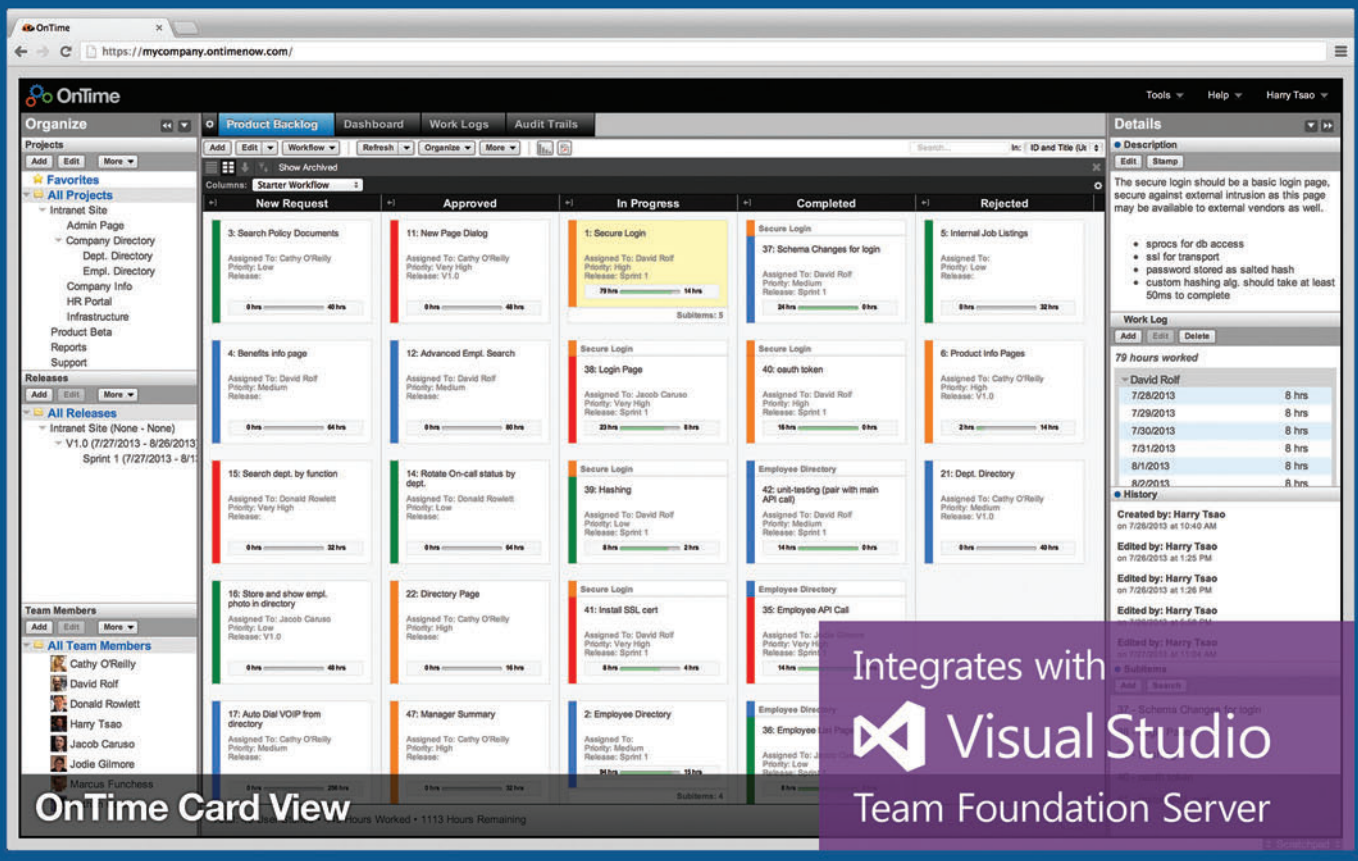
But that same availability of choice also poses a challenge. As developers shift away from big frameworks that do everything in favor of componentized stacks, they'll have to seek out the middleware to accomplish their goals. "I think that in the OWIN world—like with Node.js and Ruby—the biggest challenge will be around the paradox of choice," Dierking said, referencing Barry Schwartz's book exploring the stress produced by broad choices ("The Paradox of Choice: Why More Is Less" [Harper Perennial, 2005]).

Katana will continue to improve. Dierking said post-Katana 2.0 features will address functionality that remains locked up in the System.Web assembly such as caching and session management. Future versions will also make it easier to enable ASP.NET MVC, which remains tied to System.Web, to run in an OWIN self-hosted pipeline. There's also an effort to release new, ultra-fast servers and hosts.

Has your organization looked into adopting OWIN and Katana for Web application development? E-mail me at mmeditor@microsoft.com and tell me what you think.

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

the #1 selling scrum software



OnTime Scrum agile project management software

Want **three extra months of dev time** a year? Operate like our OnTime Scrum customers who have been able to ship their software 24% faster!

So how are they doing it? Our fast, customizable interface allows users save time and zip from one task to another on one screen—no clicking around. From there **Card View** creates a visually appealing, intuitive system for managing user stories that integrates seamlessly into Visual Studio and TFS source control—perfect for any **Microsoft development environment**.

Is it any wonder that OnTime Scrum is the **#1 selling Scrum software**?

just **$7** per user per month

Get 10% off your OnTime subscription with this link:

OnTimeNow.com/MSDN

Plus: learn about our solutions for bug tracking, help desk & customer management, and project wikis

axosoft

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

DINO ESPOSITO

Programming CSS: Bundling and Minification

An old mantra of Web development says that too many requests affect page performance. If you know a trick to reduce the number of HTTP requests triggered by your Web pages, then by all means, apply it. As Web pages get filled with richer visual content, the cost of downloading related resources such as CSS, scripts and images grows significantly. Surely, for the most part, these resources may be cached locally by the browser, but the initial footprint can be difficult to sustain. In addition, fewer and smaller requests help keep bandwidth low, reduce latency and lengthen battery life. These are critical factors in mobile browsing. A widely accepted approach to address these issues consists of a combination of two actions: bundling and minification.

In this article, I'll cover bundling and minification of CSS files from the unique perspective of software tooling available in ASP.NET MVC 4. This follows my previous column, "Creating Mobile-Optimized Views in ASP.NET MVC 4, Part 2: Using WURFL" (msdn.microsoft.com/magazine/dn342866).

Basics of Bundling and Minification

Bundling is the process of rolling up a number of distinct resources together into a single downloadable resource. For example, a bundle may consist of multiple JavaScript or CSS files you bring down to the local machine by making a single HTTP request to an ad hoc endpoint. Minification, on the other hand, is a transformation applied to a resource. In particular, minification is the removal of all unnecessary characters from a text-based resource in a way that doesn't alter the expected functionality. This means shortening identifiers, aliasing functions, and removing comments, white-space characters and new lines—in general, all characters that are usually added for readability but take up space and don't really serve any functional purpose.

Bundling and minification can be applied together but remain independent processes. Depending on the need, you can decide to only create bundles or minify individual files. Usually, however, on production sites there are no reasons not to bundle and minify all CSS and JavaScript files—except perhaps common resources such as jQuery that are likely to be on well-known content delivery networks (CDNs). At debug time, though, it's an entirely different story: A minified or bundled resource is quite hard to read and step through, so you might not want bundling and minification enabled.

Many frameworks provide bundling and minification services with slightly different levels of extensibility and different feature sets. For the most part, they all offer the same capabilities, so picking one over the other is purely a matter of preference. If you're writing an ASP.NET MVC 4 application, the natural choice for bundling and minification is the Microsoft ASP.NET Web Optimization Framework, available through a NuGet package (bit.ly/1bS8u4B) and shown in **Figure 1**.

Using CSS Bundling

As I see things, the best way to understand the mechanics of CSS bundling is to start from a truly empty ASP.NET MVC project. This means creating a new project using the Empty Project template and removing unused references and files. Next, say you add a layout file that links a couple of CSS files:

```
<link rel="stylesheet"
  href="@Url.Content("~/content/styles/site.css")"/>
<link rel="stylesheet"
  href="@Url.Content("~/content/styles/site.more.css")"/>
```

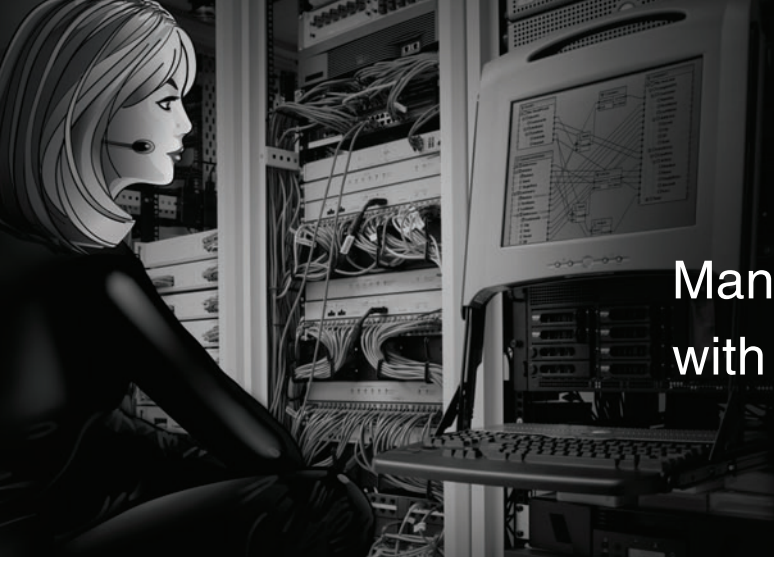
If you display the page and monitor its network activity with Fiddler or Internet Explorer Developer Tools, you see two downloads that go in parallel. This is the default behavior.

> Many frameworks provide bundling and minification services with slightly different levels of extensibility and different feature sets.

Note that in ASP.NET MVC 4 you can rewrite the previous markup in a far more compact way using the new Styles.Render facility:

```
@Styles.Render(
  "~/content/styles/site.css",
  "~/content/styles/site.more.css")
```

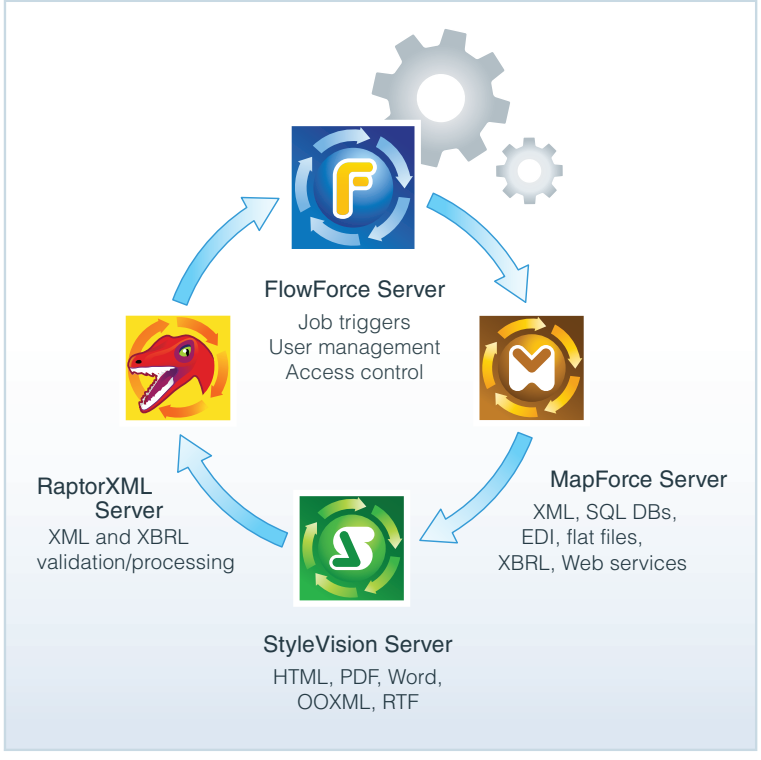
Located under System.Web.Optimization, the Styles class is far more powerful than it might seem at first. The Styles.Render method also supports bundles. This means the method has a variety of overloads, one of which accepts an array of CSS URLs. Another overload, though, takes the name of a previously created bundle (more on this shortly). In this case, it emits a single <link> element and makes it point to an auto-generated URL that returns all the style sheets as bundled or minified.



Manage Information Workflows with Altova® FlowForce® Server

ALTOVA® flowforce® SERVER 2013

Introducing FlowForce Server, the powerful new server software for managing today's multi-step, enterprise-level data validation, aggregation, processing, and reporting tasks. This flexible workflow orchestration tool provides easy-to-manage automation of essential business processes on high-performance servers.



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

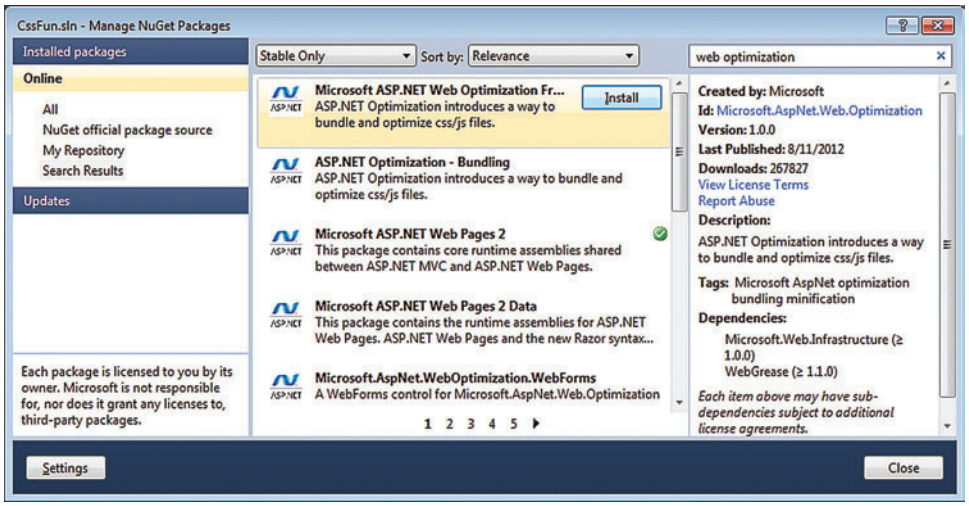


Figure 1 **Installing the Microsoft ASP.NET Web Optimization Framework**

Creating CSS Bundles

Typically, you create bundles programmatically in global.asax. In accordance with the ASP.NET MVC 4 pattern for configuration code, you might want to create a BundleConfig class under the App_Start folder and expose a static initialization method out of it:

```
BundleConfig.RegisterBundles(BundleTable.Bundles);
```

A CSS bundle is simply a collection of style sheets. Here's the code you need to group the two aforementioned CSS files in a single download:

```
public class BundleConfig
{
  public static void RegisterBundles(BundleCollection bundles)
  {
    // Register bundles first
    bundles.Add(new Bundle("~/mycss").Include(
      "~/content/styles/site.css",
      "~/content/styles/site.more.css"));

    BundleTable.EnableOptimizations = true;
  }
}
```

You create a new Bundle class and pass the constructor the virtual path that will be used to reference the bundle from within a view or Web page. You can also set the virtual path later through the Path property. To associate CSS files with the bundle, you use the Include method. This method takes an array of strings representing virtual paths. You can indicate CSS files explicitly as in the previous example or you can indicate a pattern string as shown here:

```
bundles.Add(new Bundle("~/mycss")
  .Include("~/content/styles/*.css");
```

The Bundle class also has an IncludeDirectory method that allows you to indicate the path to a given virtual directory, and possibly a pattern-matching string and a Boolean flag to also enable search on subdirectories.

The EnableOptimization Boolean property you see set on the BundleTable class in the preceding code snippet refers to the need to explicitly enable bundling. If not enabled programmatically, bundling just doesn't work. As mentioned, bundling is a form of optimization; as such, it mostly makes sense when the site is in production. The EnableOptimization property is a convenient way to set up bundling as it should be in production, but you should disable it until the site is compiled in debug mode. You might even go with the following code:

```
if (!DEBUG)
{
  BundleTable.EnableOptimizations = true;
}
```

More Advanced Bundling Features

As far as CSS bundles are concerned, there's not much else that's relevant except minification. However, the BundleCollection class is a general-purpose class that can be used also for bundling script files. In particular, the BundleCollection class has a couple of features that are worth mentioning even though they're mostly useful when script files—rather than CSS files—are bundled.

The first feature is ordering. The BundleCollection class has a property named Orderer of type IBundleOrderer. As obvious as it may seem, an orderer is a component responsible for determining the actual order in which you want files to be bundled for download. The default orderer is the DefaultBundleOrderer class. This class bundles files in the order that results from the settings set via FileSetOrderList—a property of BundleCollection. The FileSetOrderList is designed to be a collection of BundleFileSetOrdering classes. Each of these classes defines a pattern for files (for example, jquery-*), and the order in which BundleFileSetOrdering classes are added to FileSetOrderList determines the actual order of files. For example, given the default configuration, all jQuery files are always bundled before Modernizr files.

The impact of the DefaultBundleOrderer class on CSS files is more limited. If you have a file named "reset.css" or "normalize.css" in your Web site, these files are automatically bundled before any of your CSS files, and reset.css always precedes normalize.css. For those not familiar with reset/normalize style sheets, the goal of such style sheets is to provide a standard set of style attributes for all HTML elements so your pages don't inherit browser-specific settings such as fonts, sizes and margins. While some recommended content

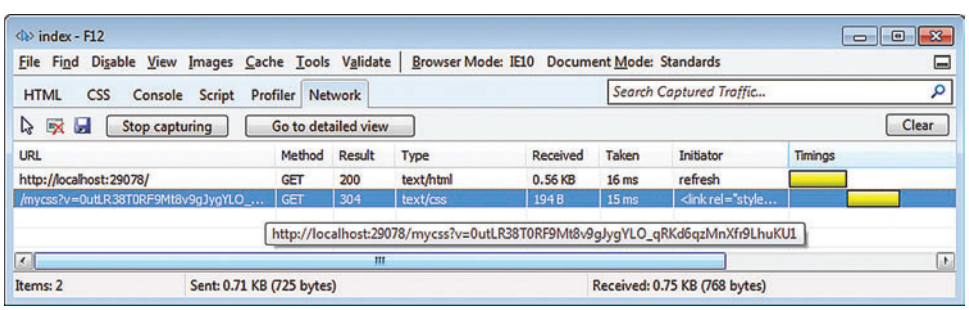


Figure 2 **The Second Request Is a Bundle with Multiple CSS Files**

telerik

deliver more than expected

Telerik DevCraft Q3

RELEASE: OCTOBER 2013

- Telerik's full stack of .NET controls and tools for the professional developer
- 350+ UI controls and reporting for all Microsoft platforms
- Productivity tools for faster coding, debugging and profiling

See what's new in Q3 2013 Release & download your 30 day free trial at

www.telerik.com

exists for both types of CSS files, the actual content is up to you. If you have files with these names in your project, then ASP.NET MVC 4 makes an extra effort to ensure that they're bundled before anything else. If you want to override the default orderer and ignore predefined bundle file set orderings, you have two options. First, you can create your own orderer that works on a per-bundle basis. Here's an example that just ignores predefined orderings:

```
public class PoorManOrderer : IBundleOrderer
{
  public IEnumerable<FileInfo> OrderFiles(
    BundleContext context, IEnumerable<FileInfo> files)
  {
     return files;
  }
}
```

You use it like so:

```
var bundle = new Bundle("~/mycss");
bundle.Orderer = new PoorManOrderer();
```

In addition, you can reset all orderings using the following code:

```
bundles.ResetAll();
```

In this case, the effect of using the default orderer or the poor man's orderer shown earlier is the same. Note, though, that ResetAll also resets script orderings.

The second, more advanced feature is the ignore list. Defined through the IgnoreList property of the BundleCollection class, it defines the pattern-matching strings for files that are selected for inclusion but should be ignored instead. The major benefit of bundles as implemented in ASP.NET MVC 4 is that you can get all JavaScript files (*.js) in the folder in a single call. However, it's likely *.js also matches files you don't want to download, such as vsdoc.js files. The default configuration for IgnoreList takes care of most common scenarios while giving you a chance to customize.

CSS Bundling is a powerful optimization feature.

CSS Bundling in Action

CSS bundling is a powerful optimization feature, but how does it work in practice? Consider the following code:

```
var bundle = new Bundle("~/mycss");
bundle.Include("~/content/styles/*.css");
bundles.Add(bundle);
```

The corresponding HTTP traffic is shown in **Figure 2**.

The first request is for the homepage; the second request doesn't point to a specific CSS file but refers to a bundle that contains all CSS files in the content/styles folder (see **Figure 3**).

Adding Minification

The Bundle class is concerned only with packing multiple resources together so they're captured in a single download and cached.

As you can see in the code in **Figure 3**, though, the downloaded content is padded with blanks and newline characters for readability purposes. However, browsers aren't concerned with readability, and for a browser, the CSS code in **Figure 3** is perfectly equivalent to the string in the following minified code:

```
html,body{font-family:'segoe ui';font-size:1.5em;margin:10px}
  html,body{background-color:#111;color:#48d1cc}
```

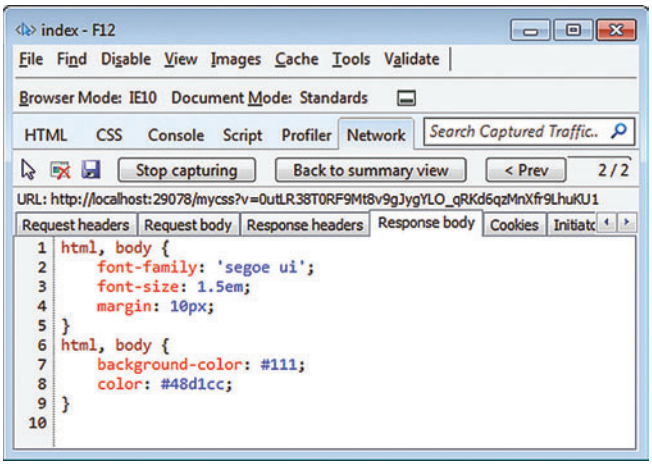


Figure 3 **An Example of Bundled CSS**

For the simple CSS I'm working with in this example, minification is probably not a big deal. However, minified CSS makes a lot of sense for business sites with large style sheets.

How would you add minification? It's as simple as replacing the Bundle class with StyleBundle. StyleBundle is surprisingly simple. It inherits from Bundle and just consists of a different constructor:

```
public StyleBundle(string virtualPath)
  : base(virtualPath, new IBundleTransform[] { new CssMinify() })
{
}
```

The Bundle class has a constructor that accepts a list of IBundleTransform objects. These transforms are applied one after the next to the content. The StyleBundle class just adds the CssMinify transformer. CssMinify is the default minifier for ASP.NET MVC 4 (and newer versions) and is based on the WebGrease optimization tools (webgrease.codeplex.com). Needless to say, if you want to switch to a different minifier, all you need to do is get the class—an implementation of IBundleTransform—and pass it via the constructor of class Bundle.

No More Excuses

The Web Optimization Framework bundled with ASP.NET MVC 4 adds a tiny bit of functionality, but it's functionality that's much better to have than not. There's simply no reason for not minifying and bundling resources. Up until now, a possible excuse was lack of native support in the ASP.NET platform. This is no longer the case with ASP.NET MVC 4 and newer versions.

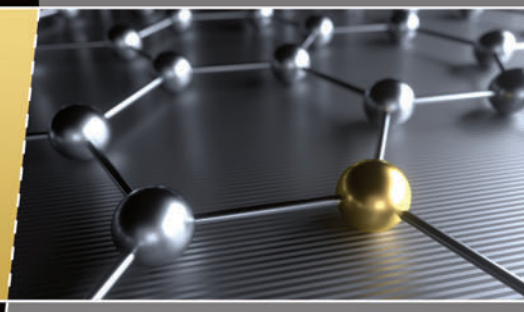
As far as CSS files are concerned, though, there's another aspect to consider: dynamic generation of styles. Designed to be a mere skin applied to pages, CSS is turning into a more dynamic resource to the point that some pseudo-programming languages have been introduced to generate CSS programmatically. Next time I'll be back to cover just that. ■

Dino Esposito *is the author of "Architecting Mobile Solutions for the Enterprise" (Microsoft Press, 2012) and the upcoming "Programming ASP.NET MVC 5" (Microsoft Press). A technical evangelist for the .NET and Android platforms at JetBrains and frequent speaker at industry events worldwide, Esposito shares his vision of software at software2cents.wordpress.com and on Twitter at twitter.com/despos.*

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

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

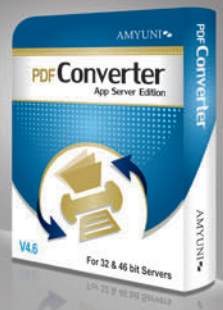
Complete Suite of Accurate PDF Components

- 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

Aspose.Total

Every Aspose component combined in one powerful suite.

Powerful File Format Components and Controls

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

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

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

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

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

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

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

... and many others!

Try Free Today!

- Aspose.Total for .NET
- Aspose.Total for Java
- Aspose.Total for SharePoint
- Aspose.Total for SSRS
- Aspose.Total for JasperReports
- Aspose.Total for Cloud

KENNY KERR

Rendering for the Windows Runtime

In my last column, I examined the Windows Runtime (WinRT) application model (msdn.microsoft.com/magazine/dn342867). I showed you how to write a Windows Store or Windows Phone app with standard C++ and classic COM, using only a handful of WinRT API functions. There's certainly no requirement that you must use a language projection such as C++/CX or C#. Being able to step around these abstractions is a powerful capability and is a great way to understand how this technology works.

My May 2013 column introduced Direct2D 1.1 and showed you how to use it to render in a desktop application (msdn.microsoft.com/magazine/dn198239). The next column introduced the dx.h library—available from dx.codeplex.com—which dramatically simplifies DirectX programming in C++ (msdn.microsoft.com/magazine/dn201741).

> In order to support rendering properly, the window must be aware of certain events.

The code in my last column was enough to bring the CoreWindow-based app to life, but it didn't provide any rendering.

This month, I'll show you how to take this basic skeleton and add support for rendering. The WinRT application model is optimized for rendering with DirectX. I'll show you how to take what you've learned in my previous columns about Direct2D and Direct3D rendering and apply it to your CoreWindow-based WinRT app—specifically using Direct2D 1.1, via the dx.h library. For the most part, the actual Direct2D and Direct3D drawing commands you'll need to write are the same regardless of whether you're targeting the desktop or the Windows Runtime. There are, however, some minor differences, and certainly getting it all hooked up in the first place is quite different. So I'll pick up where I left off last time and show you how to get some pixels on the screen!

In order to support rendering properly, the window must be aware of certain events. At a minimum, these include changes to the window's visibility and size, as well as changes to the logical display DPI configuration selected by the user. As with the Activated event I covered last time, these new events are all reported to the application via COM interface callbacks. The ICoreWindow interface provides methods to register for the VisibilityChanged and SizeChanged events, but first I need to implement the respective handlers. The two COM interfaces I need to implement are much like the Activated event handler with its Microsoft Interface Definition Language (MIDL)-generated class templates:

```
typedef ITypedEventHandler<CoreWindow *, VisibilityChangedEventArgs *>
  IVisibilityChangedEventHandler;

typedef ITypedEventHandler<CoreWindow *, WindowSizeChangedEventArgs *>
  IWindowSizeChangedEventHandler;
```

The next COM interface I must implement is called IDisplayPropertiesEventHandler, and thankfully, it's already defined. I simply need to include the relevant header file:

```
#include <Windows.Graphics.Display.h>
```

In addition, the relevant types are defined in the following namespace:

```
using namespace ABI::Windows::Graphics::Display;
```

Given these definitions, I can update the SampleWindow class from my last column to inherit from these three interfaces as well:

```
struct SampleWindow :
  ...
  IVisibilityChangedEventHandler,
  IWindowSizeChangedEventHandler,
  IDisplayPropertiesEventHandler
```

I must also remember to update my QueryInterface implementation to indicate support for these interfaces. I'll leave that for you to do. Of course, as I mentioned last time, the Windows Runtime doesn't care where these COM interface callbacks are implemented. It follows that the Windows Runtime doesn't assume that my app's IFrameworkView, the primary interface implemented by the SampleWindow class, also implements these callback interfaces. So while it's correct that QueryInterface properly handles queries for these interfaces, the Windows Runtime isn't going to query for them. Instead, I need to register for the respective events, and the best place to do so is in my implementation of the IFrameworkView Load method. As a reminder, Load is where you should stick any and all code to prepare your app for initial presentation. I can then register for the VisibilityChanged and SizeChanged events inside the Load method:

```
EventRegistrationToken token;
HR(m_window->add_VisibilityChanged(this, &token));
HR(m_window->add_SizeChanged(this, &token));
```

This tells the Windows Runtime explicitly where to find the first two interface implementations. The third and final interface is for the LogicalDpiChanged event, but this event registration is provided by the IDisplayPropertiesStatics interface. This static interface is implemented by the WinRT DisplayProperties class. I can simply use the GetActivationFactory function template to get

NEW OPPORTUNITIES WITH NEW DOMAINS

Choose from **over 700 new top-level domains**! Create a short, memorable web address that perfectly fits your business or brand, such as **your-name.blog**, **auto.shop** or **events.nyc**. You can also make your website easier to find by getting new extensions for your existing domain.

With 1&1, it is easy to connect a registered domain to any website, no matter which provider is hosting your website.

Find out more at **1and1.com**

NEW!
PRE-RESERVE
FREE
WITH NO OBLIGATION!*

1&1®

MEMBER OF united internet

DOMAINS | E-MAIL | WEB HOSTING | eCOMMERCE | SERVERS

1and1.com

* Pre-reserving a domain name does not guarantee that you will be able to register that domain. Other terms and conditions may apply. Visit www.1and1.com for full promotional offer details. Program and pricing specifications and availability are subject to change without notice. 1&1 and the 1&1 logo are trademarks of 1&1 Internet, all other trademarks are the property of their respective owners.

Figure 1 **A Dynamic Rendering Loop**

```
auto __stdcall Run() -> HRESULT override
{
  ComPtr<ICoreDispatcher> dispatcher;
  HR(m_window->get_Dispatcher(dispatcher.GetAddressOf()));

  while (true)
  {
    if (m_visible)
    {
      Render();
      HR(dispatcher->
        ProcessEvents(CoreProcessEventsOption_ProcessAllIfPresent));
    }
    else
    {
      HR(dispatcher->
        ProcessEvents(CoreProcessEventsOption_ProcessOneAndAllPending));
    }
  }

  return S_OK;
}
```

a hold of it (the implementation of GetActivationFactory can be found in my most recent column):

```
ComPtr<IDisplayPropertiesStatics> m_displayProperties;

m_displayProperties = GetActivationFactory<IDisplayPropertiesStatics>(
  RuntimeClass_Windows_Graphics_Display_DisplayProperties);
```

The member variable holds on to this interface pointer, as I'll need to call it at various points during the lifecycle of the window. For now, I can just register for the LogicalDpiChanged event inside the Load method:

```
HR(m_displayProperties->add_LogicalDpiChanged(this, &token));
```

I'll get back to the implementation of these three interfaces in a moment. Now it's time to prepare the DirectX infrastructure. I'm going to need the standard set of device resource handlers that I've discussed numerous times in previous columns:

```
void CreateDeviceIndependentResources() {}
void CreateDeviceSizeResources() {}
void CreateDeviceResources() {}
void ReleaseDeviceResources() {}
```

The first is where I can create or load any resources that aren't specific to the underlying Direct3D rendering device. The next two are for creating device-specific resources. It's best to separate resources that are specific to the window's size from those that are not. Finally, all device resources must be released. The remaining DirectX infrastructure relies on the app implementing these four methods correctly based on the app's specific needs. It provides discrete points in the app for me to manage rendering resources and the efficient creation and recycling of those resources.

Figure 2 **Outline of the Render Method**

```
void Render()
{
  if (!m_target)
  {
    // Prepare render target ...
  }

  m_target.BeginDraw();
  Draw();
  m_target.EndDraw();

  auto const hr = m_swapChain.Present();

  if (S_OK != hr && DXGI_STATUS_OCCLUDED != hr)
  {
    ReleaseDevice();
  }
}
```

Now I can bring in dx.h to take care of all the DirectX heavy lifting:

```
#include "dx.h"
```

And every Direct2D app begins with the Direct2D factory:

```
Factory1 m_factory;
```

You can find this in the Direct2D namespace, and I typically include it as follows:

```
using namespace KennyKerr;
using namespace KennyKerr::Direct2D;
```

The dx.h library has discrete namespaces for Direct2D, DirectWrite, Direct3D, Microsoft DirectX Graphics Infrastructure (DXGI) and so on. Most of my apps use Direct2D heavily, so this makes sense for me. You can, of course, manage the namespaces in whatever way makes sense for your app.

The m_factory member variable represents the Direct2D 1.1 factory. It's used to create the render target as well as a variety of other device-independent resources as needed. I'll create the Direct2D factory and then allow any device-independent resources to be created as the final step in the Load method:

```
m_factory = CreateFactory();
CreateDeviceIndependentResources();
```

After the Load method returns, the WinRT CoreApplication class immediately calls the IFrameworkView Run method.

The implementation of the SampleWindow Run method from my last column simply blocked by calling the ProcessEvents method on the CoreWindow dispatcher. Blocking in this way is adequate if your app is only going to perform infrequent rendering based on various events. Perhaps you're implementing a game or just need some high-resolution animation for your app. The other extreme is to use a continuous animation loop, but perhaps you want something a bit more intelligent. I'm going to implement something of a compromise between these two positions. First, I'll add a member variable to keep track of whether the window is visible. This will let me throttle back the rendering when the window isn't physically visible to the user:

```
bool m_visible;

SampleWindow() : m_visible(true) {}
```

I can then rewrite the Run method as shown in **Figure 1**.

As before, the Run method retrieves the CoreWindow dispatcher. It then enters an infinite loop, continuously rendering and processing any window messages (called "events" by the Windows Runtime) that may be in the queue. If, however, the window isn't visible, it blocks until a message arrives. How does the app know when the window's visibility changes? That's what the IVisibilityChangedEventHandler interface is for. I can now implement its Invoke method to update the m_visible member variable:

```
auto __stdcall Invoke(ICoreWindow *,
  IVisibilityChangedEventArgs * args) -> HRESULT override
{
  unsigned char visible;
  HR(args->get_Visible(&visible));

  m_visible = 0 != visible;
  return S_OK;
}
```

The MIDL-generated interface uses an unsigned char as a portable Boolean data type. I simply get the window's current visibility using the provided IVisibilityChangedEventArgs interface pointer

Figure 3 **Preparing the Render Target**

```
void Render()
{
  if (!m_target)
  {
    auto device = Direct3D::CreateDevice();
    m_target = m_factory.CreateDevice(device).CreateDeviceContext();

    auto dxgi = device.GetDxgiFactory();
    m_swapChain = dxgi.CreateSwapChainForCoreWindow(device, m_window.Get());
    CreateDeviceSwapChainBitmap();

    float dpi;
    HR(m_displayProperties->get_LogicalDpi(&dpi));
    m_target.SetDpi(dpi);

    CreateDeviceResources();
    CreateDeviceSizeResources();
  }

  // Drawing and presentation ... see Figure 2
```

and update the member variable accordingly. This event is raised whenever the window is hidden or shown and is a bit simpler than implementing this for desktop applications, as it takes care of a number of scenarios including app shutdown and power management, not to mention switching windows.

Next, I need to implement the Render method called by the Run method in **Figure 1**. This is where the rendering stack is created on-demand and when the actual drawing commands occur. The basic skeleton is shown in **Figure 2**.

The Render method should look familiar. It has the same basic form I've outlined before for Direct2D 1.1. It begins by creating the render target as needed. This is immediately followed by the actual drawing commands sandwiched between calls to BeginDraw and EndDraw. Because the render target is a Direct2D device context, actually getting the rendered pixels onto the screen involves presenting the swap chain. Speaking of which, I need to add the dx.h types representing the Direct2D 1.1 device context as well as the DirectX 11.1 version of the swap chain. The latter can be found in the Dxgi namespace:

```
DeviceContext m_target;
Dxgi::SwapChain1 m_swapChain;
```

The Render method concludes by calling ReleaseDevice if presentation fails:

```
void ReleaseDevice()
{
  m_target.Reset();
  m_swapChain.Reset();
  ReleaseDeviceResources();
}
```

This takes care of releasing the render target and swap chain. It also calls ReleaseDeviceResources to allow any device-specific resources such as brushes, bitmaps or effects to be released. This ReleaseDevice method might seem inconsequential, but it's critical for reliably handling device loss in a DirectX app. Without properly releasing all device resources—any resource that's backed by the GPU—your app will fail to recover from device loss and will come crashing down.

Next, I need to prepare the render target, the bit I omitted from the Render method in **Figure 2**. It starts with creating the Direct3D device (the dx.h library really simplifies the next few steps as well):

```
auto device = Direct3D::CreateDevice();
```

With the Direct3D device in hand, I can turn to the Direct2D factory to create the Direct2D device and the Direct2D device context:

```
m_target = m_factory.CreateDevice(device).CreateDeviceContext();
```

Next, I need to create the window's swap chain. I'll first retrieve the DXGI factory from the Direct3D device:

```
auto dxgi = device.GetDxgiFactory();
```

I can then create a swap chain for the application's CoreWindow:

```
m_swapChain = dxgi.CreateSwapChainForCoreWindow(device, m_window.Get());
```

Here, again, the dx.h library makes my life a lot simpler by automatically filling in the DXGI_SWAP_CHAIN_DESC1 structure for me. I'll then call out to the CreateDeviceSwapChainBitmap method to create a Direct2D bitmap that will represent the swap chain's back buffer:

```
void CreateDeviceSwapChainBitmap()
{
  BitmapProperties1 props(BitmapOptions::Target | BitmapOptions::CannotDraw,
    PixelFormat(Dxgi::Format::B8G8R8A8_UNORM, AlphaMode::Ignore));

  auto bitmap =
    m_target.CreateBitmapFromDxgiSurface(m_swapChain, props);

  m_target.SetTarget(bitmap);
}
```

This method first needs to describe the swap chain's back buffer in a way that makes sense to Direct2D. BitmapProperties1 is the dx.h version of the Direct2D D2D1_BITMAP_PROPERTIES1 structure. The BitmapOptions::Target constant indicates the bitmap will be used as the target of a device context. The BitmapOptions::CannotDraw constant relates to the fact that the swap chain's back buffer can only be used as an output and not as an input to other drawing operations. PixelFormat is the dx.h version of the Direct2D D2D1_PIXEL_FORMAT structure.

With the bitmap properties defined, the CreateBitmapFromDxgiSurface method retrieves the swap chain's back buffer and creates a Direct2D bitmap to represent it. In this way, the Direct2D device context can render directly to the swap chain simply by targeting the bitmap via the SetTarget method.

Back in the Render method, I just need to tell Direct2D how to scale any drawing commands according to the user's DPI configuration:

```
float dpi;
HR(m_displayProperties->get_LogicalDpi(&dpi));
m_target.SetDpi(dpi);
```

I'll then call out to the app's device resource handlers to create any resources as needed. To summarize, **Figure 3** provides the complete device initialization sequence for the Render method.

Although DPI scaling is properly applied immediately after the Direct2D device context is created, it also needs to be updated whenever this setting is changed by the user. The fact that DPI scaling can change for a running app is new to Windows 8. This is where the IDisplayPropertiesEventHandler interface comes in.

Figure 4 **Resizing the Swap Chain**

```
void ResizeSwapChainBitmap()
{
  m_target.SetTarget();

  if (S_OK == m_swapChain.ResizeBuffers())
  {
    CreateDeviceSwapChainBitmap();
    CreateDeviceSizeResources();
  }
  else
  {
    ReleaseDevice();
  }
}
```

I can now simply implement its Invoke method and update the device accordingly. Here's the LogicalDpiChanged event handler:

```
auto __stdcall Invoke(IInspectable *) -> HRESULT override
{
  if (m_target)
  {
    float dpi;
    HR(m_displayProperties->get_LogicalDpi(&dpi));
    m_target.SetDpi(dpi);
    CreateDeviceSizeResources();
    Render();
  }

  return S_OK;
}
```

Assuming the target—the device context—has been created, it retrieves the current logical DPI value and simply forwards it on to Direct2D. It then calls out to the app to recreate any device-size-specific resources before re-rendering. In this way, my app can dynamically respond to changes in the display's DPI configuration. The final change that the window must handle dynamically is that of changes to the window's size. I've already hooked up the event registration, so I simply need to add the implementation of the IWindowSizeChangedEventHandler Invoke method, representing the SizeChanged event handler:

```
auto __stdcall Invoke(ICoreWindow *,
  IWindowSizeChangedEventArgs *) -> HRESULT override
{
  if (m_target)
  {
    ResizeSwapChainBitmap();
    Render();
  }

  return S_OK;
}
```

The only thing left to do is resize the swap chain bitmap via the ResizeSwapChainBitmap method. Again, this is something that needs to be handled carefully. Resizing the swap chain's buffers can and should be an efficient operation, but only if done correctly. First, in order for this operation to succeed, I need to ensure all references to these buffers have been released. These may be references the app is holding directly as well as indirectly. In this case, the reference is held by the Direct2D device context. The target image is the Direct2D bitmap I created to wrap the swap chain's back buffer. Releasing this is easy enough:

```
m_target.SetTarget();
```

> The fact that DPI scaling can change for a running app is new to Windows 8.

I can then call the swap chain's ResizeBuffers method to do all of the heavy lifting, and then call out to the app's device resource handlers as needed. **Figure 4** shows you how this comes together.

You can now add some drawing commands, and they'll be rendered efficiently by DirectX to the target of the CoreWindow. As a simple example, you might want to create a solid-color brush inside the CreateDeviceResources handler and assign it to a member variable as follows:

```
SolidColorBrush m_brush;

m_brush = m_target.CreateSolidColorBrush(Color(1.0f, 0.0f, 0.0f));
```

Inside the window's Draw method, I can start by clearing the window's background with white:

```
m_target.Clear(Color(1.0f, 1.0f, 1.0f));
```

I can then use the brush and draw a simple red rectangle as follows:

```
RectF rect (100.0f, 100.0f, 200.0f, 200.0f);
m_target.DrawRectangle(rect, m_brush);
```

To ensure the app can gracefully recover from device loss, I must ensure that it releases the brush at just the right time:

```
void ReleaseDeviceResources()
{
  m_brush.Reset();
}
```

And that's all it takes to render a CoreWindow-based app with DirectX. Of course, if you compare this to my May 2013 column, you should be pleasantly surprised by how much simpler the DirectX-related code works out to be thanks to the dx.h library. But as it stands, there's still a good deal of boilerplate code, mainly related to implementing the COM interfaces. This is where C++/CX comes in and simplifies the use of WinRT APIs inside your apps. It hides some of the boilerplate COM code that I've shown you in these last two columns. ■

Kenny Kerr *is a computer programmer based in Canada, an author for Pluralsight and a Microsoft MVP. He blogs at kennykerr.ca and you can follow him on Twitter at twitter.com/kennykerr.*

JULIE LERMAN

Coding for Domain-Driven Design: Tips for Data-Focused Devs, Part 3

This is the final installment of my series on helping data-focused developers wrap their heads around some of the more challenging coding concepts used with Domain-Driven Design (DDD). As a Microsoft .NET Framework developer using Entity Framework (EF), and with a long history of data-first (and even database-first) development, I've struggled and argued and whined my way to understanding how to merge my skills with some of the implementation techniques of DDD. Even if I'm not using a full DDD implementation (from client interaction all the way to the code) in a project, I still benefit greatly from many of the tools of DDD.

In this last installment, I'll discuss two important technical patterns of DDD coding and how they apply to the object-relational mapping (ORM) tool I use, EF. In an earlier installment I talked about one-to-one relationships. Here, I'll explore unidirectional relationships—preferred with DDD—and how they affect your application. This choice leads to a difficult decision: recognizing when you might be better off without some of the nice relationship "magic" that EF performs. I'll also talk a bit about the importance of balancing tasks between an aggregate root and a repository.

Build Unidirectional Relationships from the Root

From the time I started building models with EF, two-way relationships have been the norm, and I did this without thinking too hard about it. It makes sense to be able to navigate in two directions. If you have orders and customers, it's nice to be able to see the orders for a customer and, given an order, it's convenient to access the customer data. Without thinking, I also built a two-way relationship between orders and their line items. A relationship from order to line items makes sense. But if you stop to consider this for a moment, the scenarios in which you have a line item and need to get back to its order are few and far between. One that I can think of is that you're reporting on products and want to do some analysis on what products are commonly ordered together, or an analysis that involves customer or shipping data. In such cases, you might need to navigate from a product to the line items in which it's contained and then back to the order. However, I can see this coming up only in a reporting scenario where I'm not likely needing to work with the DDD-focused objects.

Code download available at archive.msdn.microsoft.com/mag201310DataPoints.

If I only need to navigate from order to line items, what is the most efficient way to describe such a relationship in my model?

As I noted, DDD prefers unidirectional relationships. Eric Evans advises that "it's important to constrain relationships as much as possible," and that "understanding the domain may reveal natural directional bias." Managing the complexities of relationships—especially when you're depending on the Entity Framework to maintain associations—is definitely an area that can cause a lot of confusion. I've already penned a number of Data Points columns devoted to associations in Entity Framework. Any level of complexity that can be removed is probably beneficial.

Contemplating the simple sales model I've used for this series on DDD, it does present a bias in the direction of an order to its line items. I can't imagine creating, deleting or editing a line item without starting from the order.

If you look back at the Order aggregate I built earlier in the series, the order does control the line items. For example, you need to use the CreateLineItem method of the Order class to add a new line item:

```
Public void CreateLineItem(Product product, int quantity)
{
  var item = new LineItem
  {
    OrderQty = quantity,
    ProductId = product.ProductId,
    UnitPrice = product.ListPrice,
    UnitPriceDiscount = CustomerDiscount + PromoDiscount
  };
  LineItems.Add(item);
}
```

The LineItem type has an OrderId property, but no Order property. That means it's possible to set the value of OrderId, but you can't navigate from a LineItem to an actual Order instance.

In this case, I have, in Evans' words, "imposed a traversal direction." I have, in effect, ensured I can traverse from Order to LineItem but not in the other direction.

There are implications to this approach not only in the model but also in the data layer. I use Entity Framework as my ORM tool and it comprehends this relationship well enough simply from the LineItems property of the Order class. And because I happen to follow the conventions of EF, it understands that LineItem.OrderId is my foreign key property back to the Order class. If I used a different name for OrderId, things would be more complicated for Entity Framework.

But in this scenario, I can add a new LineItem to an existing order like this:

```
order.CreateLineItem(aProductInstance, 2);
var repo = new SimpleOrderRepository();
repo.AddAndUpdateLineItemsForExistingOrder(order);
repo.Save();
```

Visual Studio 2012 Ready

With royalty-free licensing, create:

- Form-based reports, such as invoices & insurance documents
- Transaction reports, such as sales & accounting
- Analytical reports, such as sales & budget analysis & portfolio analysis

ActiveReports 7

Download your free trial @
www.componentone.com

ComponentOne®
a division of GrapeCity®

The order variable now represents a graph with a preexisting order and a single new LineItem. That preexisting order has come from the database and already has a value in OrderId, but the new LineItem has only the default value for its OrderId property, and that's 0.

My repository method takes that order graph, adds it to my EF context and then applies the proper state, as shown in **Figure 1**.

In case you aren't familiar with EF behavior, the Add method causes the context to begin tracking everything in the graph (the order and the single line item). At the same time, each object in the graph is flagged with the Added state. But because this method is focused on using a preexisting order, I know that Order is not new and, therefore, the method fixes the state of the Order instance by setting it to Unchanged. It also checks for any preexisting LineItems and sets their state to Modified so they'll be updated in the database rather than inserted as new. In a more fleshed-out application, I'd use a pattern for more definitively knowing the state of each object, but I don't want this sample to get bogged down with additional details. (You can see an early version of this pattern on Rowan Miller's blog at bit.ly/1cLoo14, and an updated example in our coauthored book "Programming Entity Framework: DbContext" [O'Reilly Media, 2012].)

Because all of these actions are being done while the context is tracking the objects, Entity Framework also "magically" fixes the value of the OrderId in my new LineItem instance. Therefore, by the time I call Save, the LineItem knows that the OrderId value is 1.

Letting Go of the EF Relationship-Management Magic—for Updates

This good fortune occurs because my LineItem type happens to follow EF convention with the foreign key name. If you named it something other than OrderId, such as OrderFK, you'd have to make some changes to your type (for example, introducing the unwanted Order navigation property) and then specify EF mappings. This isn't desirable, as you'd be adding complexity simply to satisfy the ORM. Sometimes that may be necessary, but when it's not I prefer to avoid it.

It would be simpler to just let go of any dependency on the EF relationship magic and control the setting of the foreign key in your code.

The first step is to tell EF to ignore this relationship; otherwise, it will continue to look for a foreign key.

Here's code I'll use in the DbContext.OnModelBuilder method override so that EF won't pay attention to that relationship:

```
modelBuilder.Entity<Order>().Ignore(o => o.LineItems);
```

Now, I'll take control of the relationship myself. This means refactoring so I add a constructor to LineItem that requires OrderId and other values, and it makes LineItem much more like a DDD entity so I'm happy. I also have to modify the CreateLineItem method in Order to use that constructor rather than an object initializer.

Figure 2 shows an updated version of the repository method.

Notice I'm no longer adding the order graph and then fixing the order's state to Unchanged. In fact, because EF is unaware of the relationship, if I called context.Orders.Add(order), it would add the order instance but wouldn't add the related line items as it did before.

Instead, I'm iterating through the graph's line items and not only setting the state of existing line items to Modified but setting the state of new ones to Added. The DbContext.Entry syntax I'm using does two things. Before it sets the state, it checks to see if the context is already aware of (or "tracking") that particular entity. If it's not, then internally it attaches the entity. Now it's able to respond to the fact that the code is setting the state property. So in that single line of code, I'm attaching and setting the state of the LineItem.

My code is now in accord with another healthy prescription for using EF with DDD, which is: don't rely on EF to manage relationships. EF performs a lot of magic, a huge bonus in many scenarios. I've happily benefited from this for years. But for DDD aggregates, you really want to manage those relationships in your model and not rely on the data layer to perform necessary actions for you.

Because I'm stuck for the time being using integers for my keys (Order.OrderId, for example) and depending on my database to provide the values of those keys, I need to do some extra work in the repository for new aggregates such as a new order with line items. I'll need tight control of the persistence so I can use the old-fashioned pattern of inserting graphs: insert order, get new database-generated OrderId value, apply that to the new line items, and save them to the database. This is necessary because I've broken the relationship that EF would normally use to perform this magic. You can see in the sample download how I've implemented this in the repository.

I'm ready, after many years, to stop depending on the database to create my identifier and begin to use GUIDs for my key values, which I can generate and assign in my app. This allows me to further separate my domain from the database.

Keeping the EF Relationship-Management Magic—for Queries

Divesting my model of EF relationships really helped in the previous scenario for performing updates. But I don't want to lose all of the relationship features of EF. Loading related data when querying from the database is one feature I don't want to give up. Whether I'm eager loading, lazy loading or explicitly loading, I love benefiting from the ability of EF to bring related data along without having to express and execute additional queries.

This is where an extended view of the separation of concerns concept comes into play. When following DDD precepts for design, it's not unusual to have different representations of similar classes. For example, you might do this with a Customer class designed to be used in the context of customer management, as opposed to a Customer class for simply populating a pick list that needs only the customer's name and identifier.

Figure 1 **Applying State to an Order Graph**

```
Public void AddAndUpdateLineItemsForExistingOrder(Order order)
{
_context.Orders.Add(order);
_context.Entry(order).State = EntityState.Unchanged;
foreach (var item in order.LineItems)
{
  // Existing items from database have an Id & are being modified, not added
  if (item.LineItemId > 0)
  {
    _context.Entry(item).State = EntityState.Modified;
  }
}
}
```

Spread Studio for .NET contains our new cross-platform spreadsheet controls for Windows Forms, ASP.NET, WPF, WinRT, and Silverlight in one amazing package.

Experience for yourself:

Excel®-like grid. Native Microsoft® Excel® Compatibility - same look & feel for your users

Flexible & familiar spreadsheet architecture, advanced charting & powerful formula library

Create financial modeling & risk analysis, budgeting, insurance, scientific applications & more

Spread Studio 7 for .NET

Download your free trial @
www.componentone.com

ComponentOne®
a division of GrapeCity®

It also makes sense to have different DbContext definitions. In scenarios where you're retrieving data, you might want a context that's aware of the relationship between Order and LineItems so you can eagerly load an order along with its line items from the database. But then, when you're performing updates as I did earlier, you may want a context that explicitly ignores that relationship so you can have more granular control of your domain.

An extreme view of this for a certain subset of complex problems you may be solving with software is a pattern called Command Query Responsibility Segregation (CQRS). CQRS guides you to think of data retrieval (reads) and data storage (writes) as separate systems that may require distinct models and architectures. My small example, which highlights the benefit of having the data-retrieval operations embrace a different understanding of relationships than data-storage operations, gives you an idea of what CQRS can help you achieve. You can learn more about CQRS from the excellent resource, CQRS Journey, available at msdn.microsoft.com/library/jj554200.

Data Access Happens in the Repository, Not the Aggregate Root

I want to back up a bit now and tackle one last question that gnawed at me when I started focusing on unidirectional relationships. (This is not to say that I have no more questions about DDD, but this is the final topic I'll address in this series.) This question about unidirectional relationships is a common one for us "database-first" thinkers: Where, exactly (with DDD), does data access take place?

When EF was first released, the only way it could work with a database was to reverse-engineer an existing database. So, as I noted earlier, I got used to every relationship being two-way. If the Customers and Orders tables in the database had a primary key/foreign key constraint describing a one-to-many relationship, I saw that one-to-many relationship in the model. Customer had a navigation property to a collection of orders. Order had a navigation property to an instance of Customer.

As things evolved to Model- and Code-First, where you can describe the model and generate a database, I continued to follow that pattern, defining navigation properties on both ends of a relationship. EF was happy, mappings were simpler and coding was more natural.

So, with DDD, when I found myself with an Order aggregate root that was aware of CustomerId or maybe even a full Customer type, but I couldn't navigate from Order back to Customer, I got upset. The first question I asked was, "what if I want to find all of the orders for a customer?" I always assumed I'd need to be able to do that, and I was used to relying on having access to navigation in both directions.

If logic begins with my order aggregate root, how would I ever answer that question? I also initially had the misconception that you do everything through the aggregate root, which didn't help.

The solution made me hit my head and feel a bit foolish. I share my foolishness here in case someone else gets stuck in the same way. It's not the job of the aggregate root, nor the job of the Order, to help me answer that question. However, in an Order-focused repository, which is what I'd use to perform my queries and persistence, there's no reason I can't have a method to answer my question:

```
public List<Order>GetOrdersForCustomer(Customer customer)
  {
    return _context.Orders.
      Where(o => o.CustomerId == customer.Id)
      .ToList();
  }
```

The method returns a list of Order aggregate roots. Of course, if I'm creating this in the scope of doing DDD, I'd only bother putting that method in my repository if I know it's going to be needed in the particular context, not "just in case." Chances are, I'd need it in a reporting app or something similar, but not necessarily in a context designed for building sales orders.

Only the Beginning of My Quest

As I've learned about DDD over the past few years, the topics I covered in this series are the ones that I had the most difficulty either comprehending or figuring out how to implement when Entity Framework would be part of my data layer. Some of the frustration I encountered was due to years of thinking about my software from the perspective of how things would work in my database. Letting go of this perspective has been freeing because it lets me focus on the problem at hand—the domain problem for which I'm designing software. At the same time, I do need to find a healthy balance because there may be data-layer issues I encounter when it's time to add that into my solution.

While I've focused on how things might work when I'm mapping my classes directly back to the database with Entity Framework, it's important to consider that there could be another layer (or more) between the domain logic and the database. For example, you might have a service with which your domain logic interacts. At that point, the data layer is of little (or no) consequence to mapping from your domain logic; that problem now belongs to the service.

There are many ways to approach your software solutions. Even when I'm not implementing a full end-to-end DDD approach (something that takes quite a bit of mastery), my entire process continues to benefit from the lessons and techniques I'm learning from DDD. ■

Figure 2 **The Repository Method**

```
Public void UpdateLineItemsForExistingOrder(Order order)
{
  foreach (var item in order.LineItems)
  {
    if (item.LineItemId > 0)
    {
      _context.Entry(item).State = EntityState.Modified;
    }
    else
    {
      _context.Entry(item).State = EntityState.Added;
      item.SetOrderIdentity(order.OrderId);
    }
  }
}
```

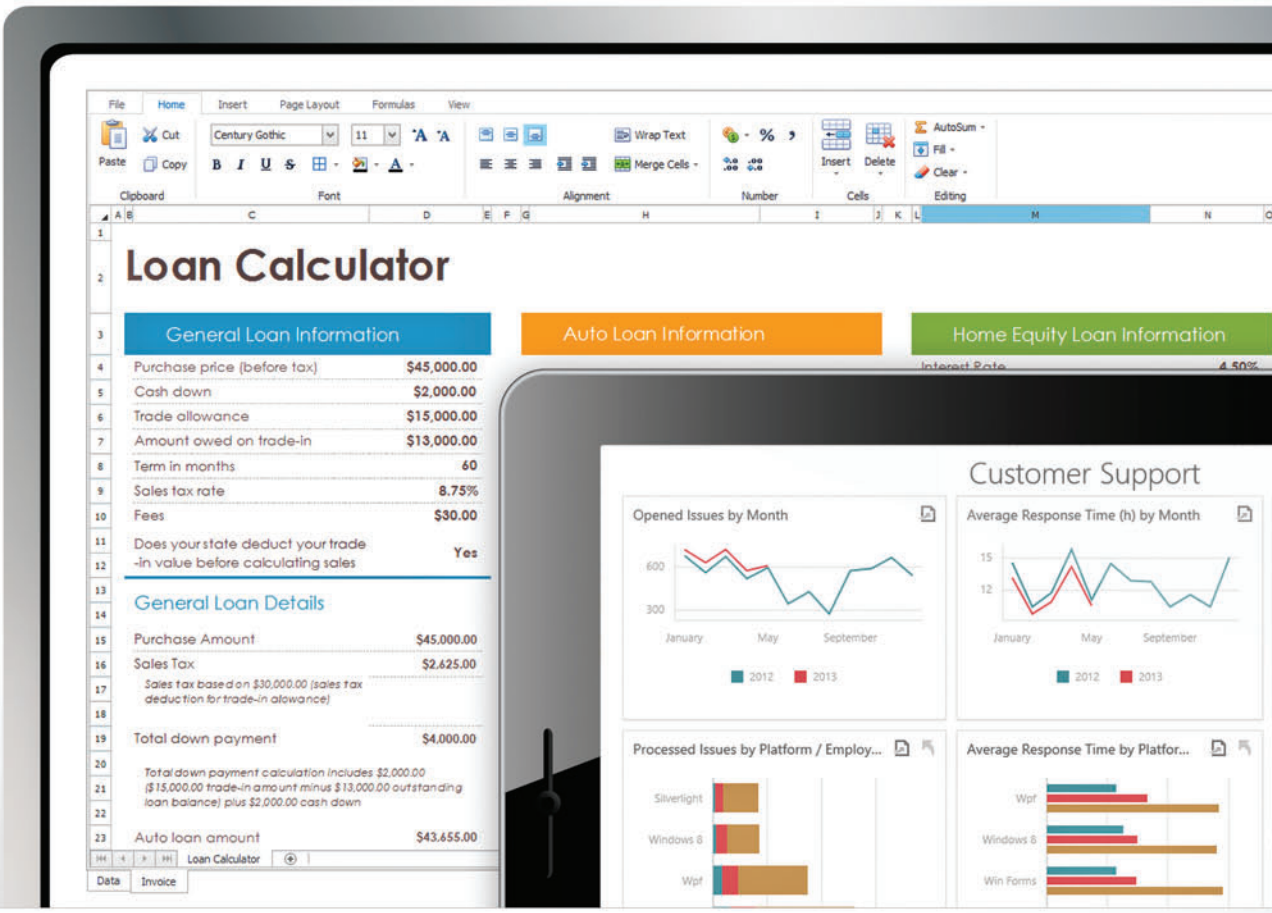
Julie Lerman *is a Microsoft MVP, .NET mentor and consultant who lives in the hills of Vermont. You can find her presenting on data access and other Microsoft .NET Framework topics at user groups and conferences around the world. She blogs at thedatafarm.com/blog and is the author of "Programming Entity 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 technical expert for reviewing this article: *Stephen Bohlen (Microsoft)*

DevExpress™

When only the best will do.

High-Performance, Elegant and Feature-Complete .NET Controls and Libraries



Download your FREE 30-day trial today. **DevExpress.com/try**

WIN WinForms · ASP ASP.NET · WPF WPF · SL Silverlight · Windows 8 XAML · HTML JS

LIVE EVENT

20 LIVE! CELEBRATING TWENTY YEARS

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

WE'VE GOT YOUR TICKET TO CODE!

INTENSE TAKE-HOME TRAINING FOR DEVELOPERS, SOFTWARE ARCHITECTS AND DESIGNERS

Celebrating 20 years of education and training for the developer community, Visual Studio Live! returns to Orlando – and we've got your ticket to Code! Visual Studio Live! Orlando is where developers, software architects and designers will connect for five days of unbiased and cutting-edge education on the Microsoft platform.

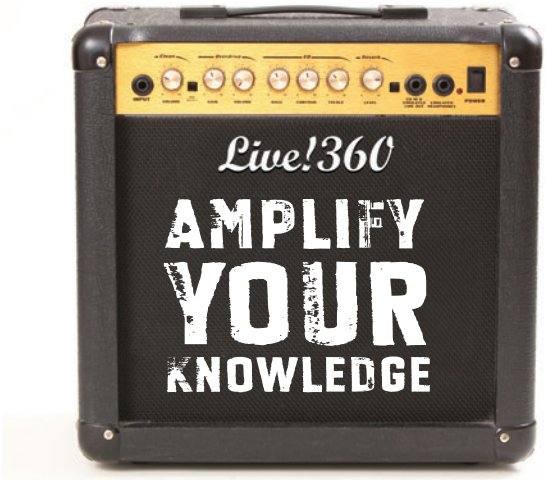
SUPPORTED BY Microsoft Visual Studio msdn magazine Visual Studio MAGAZINE

PRODUCED BY 1105 MEDIA

YOUR BACKSTAGE PASS TO THE MICROSOFT PLATFORM

ORLANDO | NOVEMBER 18-22, 2013

Royal Pacific Resort at Universal Orlando



LIVE! 360
IT EVENTS WITH PERSPECTIVE

WHETHER YOU ARE AN:

- Developer
- Programmer
- Software Architect
- Software Designer

You will walk away from this event having expanded your .NET skills and the ability to build better applications.

Visual Studio Live! Orlando is part of Live! 360, the ultimate IT and Developer line-up. This means you'll have access to four (4) events, 20 tracks, and hundreds of sessions to choose from – mix and match sessions to create your own, custom event line-up – it's like no other conference available today!

REGISTER BY OCTOBER 9 AND SAVE $300!

Use Promo Code ORLOCT4

Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS

SQL Server LIVE!
TRAINING FOR DBAs AND IT PROS

SharePoint LIVE!
TRAINING FOR COLLABORATION

Modern Apps LIVE!
MODERN APPS FROM START TO FINISH

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

TURN THE PAGE FOR MORE EVENT DETAILS

ORLANDO | **NOVEMBER 18-22, 2013**

Royal Pacific Resort at Universal Orlando

AGENDA AT-A-GLANCE

Windows 8.1 / WinRT	Web and JavaScript Development	Visual Studio / .NET Framework	Azure / Cloud Computing	Mobile Development	Data Management

START TIME	END TIME	Visual Studio Live! Pre-Conference: Sunday, November 17, 2013
4:00 PM	9:00 PM	Pre-Conference Registration - Royal Pacific Resort Conference Center
6:00 PM	9:00 PM	Dine-A-Round Dinner @ Universal CityWalk

START TIME	END TIME	Visual Studio Live! Pre-Conference Workshops: Monday, November 18, 2013 (Separate entry fee required)		
6:30 AM	8:00 AM	Pre-Conference Workshop Registration - Coffee and Morning Pastries		
8:00 AM	5:00 PM	**VSM01** - Build an Application in a Day on Windows 8 - *Philip Japikse*	**VSM02** - Rich Data HTML Mobile and Browser Clients with Knockout, JQuery, Breeze, and Web API - *Brian Noyes*	**VSM03** - End-to-End Service Orientation: Designing, Developing, & Implementing Using WCF and the Web API - *Miguel Castro*
5:00 PM	7:00 PM	EXPO Preview		
7:00 PM	8:00 PM	Live! 360 Keynote: To Be Announced		

START TIME	END TIME	Visual Studio Live! Day 1: Tuesday, November 19, 2013			
7:00 AM	8:00 AM	Registration - Coffee and Morning Pastries			
8:00 AM	9:00 AM	Visual Studio Live! Keynote: To Be Announced			
9:00 AM	9:30 AM	Networking Break • Visit the EXPO			
9:30 AM	10:45 AM	**VST01** -What's New in Windows 8.1 for Developers - *Brian Peek*	**VST02** - Patterns for JavaScript - *John Papa*	**VST03** - Overview and What's New in Windows Azure - *Eric D. Boyd*	**VST04** - What's New in Visual Studio 2013? - *Brian Noyes*
11:00 AM	12:15 PM	**VST05** - Controlling Hardware Using Windows 8.1 - *Brian Peek*	**VST06** - Jump-Start for Building Single Page Apps - *John Papa*	**VST07** - IaaS in Windows Azure with Virtual Machines - *Eric D. Boyd*	**VST08** - What's New in the .NET 4.5 BCL - *Jason Bock*
12:15 PM	2:00 PM	Lunch • Visit the EXPO			
2:00 PM	3:15 PM	**VST09** - A Lap Around Windows Phone 8 Development - *David Isbitski*	**VST10** - WCF & Web API: Can We All Just Get Along?!? - *Miguel Castro*	**VST11** - Solving Security and Compliance Challenges with Hybrid Clouds - *Eric D. Boyd*	**VST12** - How to Be a C# Ninja in 10 Easy Steps - *Benjamin Day*
3:15 PM	4:15 PM	Networking Break • Visit the EXPO			
4:15 PM	5:30 PM	**VST13** - Developing a Modern Mobile App Strategy - *Todd Anglin*	**VST14** - Building Rich Data HTML Client Apps with Breeze.js - *Brian Noyes*	**VST15** - Cloud Connected Apps with Azure Mobile Services - *David Isbitski*	**VST16** - Software Testing with Visual Studio 2013 and Team Foundation Server 2013 - *Benjamin Day*
5:30 PM	7:30 PM	Exhibitor Reception			

START TIME	END TIME	Visual Studio Live! Day 2: Wednesday, November 20, 2013			
7:00 AM	8:00 AM	Registration - Coffee and Morning Pastries			
8:00 AM	9:00 AM	Live! 360 Keynote: To Be Announced			
9:15 AM	10:30 AM	**VSW01** - What's New in WPF 4.5? - *Walt Ritscher*	**VSW02** - Slice Development Time with ASP.NET MVC, Visual Studio, and Razor - *Philip Japikse*	**VSW03** - Build the Next YouTube: Windows Azure Media Services - *Sasha Goldshtein*	**VSW04** - NoSQL for the SQL Guy - *Ted Neward*
10:30 AM	11:00 AM	Networking Break • Visit the EXPO			
11:00 AM	12:15 PM	**VSW05** - Learning UX Design Principles Through Windows 8 Apps - *Billy Hollis*	**VSW06** - Doing More with LESS for CSS - *Todd Anglin*	**VSW07** - JavaScript, Meet Cloud: Node.js on Windows Azure - *Sasha Goldshtein*	**VSW08** - Gaining the Knowledge of the Open Data Protocol (OData) - *Christopher Woodruff*
12:15 PM	1:45 PM	Round Table Lunch • Visit the EXPO			
1:45 PM	3:00 PM	**VSW09** - Building Windows Store Business Apps with Prism - *Brian Noyes*	**VSW10** - Building Mobile Cross-Platform Apps with HTML5 & PhoneGap - *Nick Landry*	**VSW11** - Applied Windows Azure - *Vishwas Lele*	**VSW12** - Seeking Life Beyond Relational: RavenDB - *Sasha Goldshtein*
3:00 PM	4:00 PM	Networking Break • Visit the EXPO • Expo Raffle @ 3:30 p.m.			
4:00 PM	5:15 PM	**VSW13** - Migrating from WPF to WinRT - *Rockford Lhotka*	**VSW14** - Connecting to Data from Windows Phone 8 - *Christopher Woodruff*	**VSW15** - Windows Azure in the Enterprise: Hybrid Scenarios - *Vishwas Lele*	**VSW16** - Session To Be Announced
8:00 PM	10:00 PM	Live! 360 Evening Event			

START TIME	END TIME	Visual Studio Live! Day 3: Thursday, November 21, 2013			
7:00 AM	8:00 AM	Registration - Coffee and Morning Pastries			
8:00 AM	9:15 AM	**VSH01** - Windows 8 HTML/JS Apps for the ASP.NET Developer - *Adam Tuliper*	**VSH02** - Create Data Driven Web Sites with WebMatrix 3 and ASP.NET Web Pages - *Rachel Appel*	**VSH03** - Making the Most of the TFS Service - *Brian Randell*	**VSH04** - iOS Development Survival Guide for the .NET Guy - *Nick Landry*
9:30 AM	10:45 AM	**VSH05** - Interaction and Navigation Patterns in Windows 8 Apps - *Billy Hollis*	**VSH06** - Building Real-Time, Multi-User Interactive Web and Mobile Applications Using SignalR - *Marcel de Vries*	**VSH07** - Continuous Integration Builds and TFS - *Brian Randell*	**VSH08** - iOS Development Survival Guide for the .NET Guy, Part 2 - *Nick Landry*
11:00 AM	12:15 PM	**VSH09** - Enhancing the Windows 8 Start Screen with Tiles, Toast and Notifications - *Walt Ritscher*	**VSH10** - Build Data-Driven Mobile Web Apps with ASP.NET MVC and jQuery Mobile - *Rachel Appel*	**VSH11** - IntelliTrace, What is it and How Can I Use it to My Benefit? - *Marcel de Vries*	**VSH12** - Session To Be Announced
12:15 PM	1:30 PM	Lunch			
1:30 PM	2:45 PM	**VSH13** - Adventures in Unit Testing: TDD vs. TED - *Ben Hoelting*	**VSH14** - Maximizing Entity Framework 6 in Your Web Projects - *Adam Tuliper*	**VSH15** - Modern .NET Development Practices and Principles - *Jason Bock*	**VSH16** - Sharing Up to 80% of Code Building Mobile Apps for iOS, Android, WP 8 and Windows 8 - *Marcel de Vries*
3:00 PM	4:15 PM	**VSH17** - Blend for Visual Studio: Design Your XAML or HTML5\CSS3 UI Faster - *Ben Hoelting*	**VSH18** - Patterns and Tools for Parallel Programming - *Marcel de Vries*	**VSH19** - Static Analysis in .NET - *Jason Bock*	**VSH20** - Talk to Me. Using Speech in Your Windows Phone App - *Walt Ritscher*
4:30 PM	5:45 PM	Live! 360 Conference Wrap-up			

START TIME	END TIME	Visual Studio Live! Post-Conference Workshops: Friday, November 22, 2013 (Separate entry fee required)	
7:00 AM	8:00 AM	Post-Conference Workshop Registration - Coffee and Morning Pastries	
8:00 AM	5:00 PM	**VSF01** - Workshop: NoSQL - *Ted Neward*	**VSF02** - Workshop: Visual Studio ALM—To Infinity and Beyond - *Brian Randell*

**Speakers and sessions subject to change*

Getting Started with the Katana Project

Howard Dierking

When ASP.NET was first released back in 2002, times were different. The Internet was still in a state of relative infancy, with around 569 million users spending an average of 46 minutes a day on about 3 million Web sites. These same measurements taken just 10 years later reveal an Internet of approximately 2.27 billion users spending a daily average of 4 hours on 555 million Web sites (see bit.ly/MY7GzO).

Clearly, that growth spurred corresponding changes in the needs of application developers in terms of the underlying frameworks, tools, and runtimes they use to build and run Web applications. Modern Web apps need to be able to evolve rapidly, taking advantage of many different components and frameworks—and they need a small footprint in order to run efficiently in the large-scale runtime of the cloud.

The Katana project is still in prerelease and is based on Visual Studio 2013 Preview; all information is subject to change.

This article discusses:

- The Open Web Interface for .NET (OWIN)
- The Katana project
- Building a composite OWIN application with a variety of different frameworks

Technologies discussed:

ASP.NET, Open Web Interface for .NET (OWIN), SignalR, Visual Studio 2013 Preview

Code download available at:

bit.ly/1alOF4m

Ensuring ASP.NET can meet these current and future needs is the core motivation behind the Katana project.

What Is Katana?

The seeds of the Katana project were actually sown outside of Microsoft with an open source project called the Open Web Interface for .NET (OWIN), a specification that defines the interactions between Web servers and application components (see owin.org). Because the goal of the specification is to stimulate a broad and vibrant ecosystem of Microsoft .NET Framework-based Web servers and application components, it reduces all interactions between servers and applications to a small set of types and a single function signature known as the application delegate, or AppFunc:

```
using AppFunc = Func<IDictionary<string, object>, Task>;
```

Every component in an OWIN-based application supplies an application delegate to a server. The components are then chained together into a pipeline into which an OWIN-based server pushes requests. In order to use resources efficiently, all components in the pipeline should be asynchronous, and this is reflected in the application delegate returning a Task object.

All states, including application state, request state, server state and so forth, are held in the IDictionary<string, object> object specified on the application delegate. This data structure, known as the environment dictionary, is passed from component to component as a request progresses through the pipeline. While any key/value data may be inserted into the environment dictionary, the OWIN specification defines keys for some core elements of HTTP, as shown in **Figure 1**.

Defining a base set of environment dictionary key-value pairs enables many different framework and component authors to interoperate in an OWIN pipeline, without forcing agreement on a specific .NET object model, such as those of HttpContextBase in ASP.NET MVC or HttpRequestMessage/HttpResponseMessage in ASP.NET Web API.

These two elements—the application delegate and the environment dictionary—form the OWIN specification. The Katana

Figure 1 **Required Environment Dictionary Keys for an HTTP Request**

Key Name	Value Description
`"owin.RequestBody"`	A Stream with the request body, if any. Stream.Null can be used as a placeholder if there's no request body.
`"owin.RequestHeaders"`	An IDictionary<string, string[]> of request headers.
`"owin.RequestMethod"`	A string containing the HTTP request method of the request (such as GET and POST).
`"owin.RequestPath"`	A string containing the request path. The path must be relative to the "root" of the application delegate.
`"owin.RequestPathBase"`	A string containing the portion of the request path corresponding to the "root" of the application delegate.
`"owin.RequestProtocol"`	A string containing the protocol name and version (such as HTTP/1.0 or HTTP/1.1).
`"owin.RequestQueryString"`	A string containing the query string component of the HTTP request URI, without the leading "?" (such as foo=bar&baz=quux). The value may be an empty string.
`"owin.RequestScheme"`	A string containing the URI scheme used for the request (such as HTTP or HTTPS).

project is the set of OWIN-based components and frameworks created and shipped by Microsoft.

Katana components can be viewed through the architectural stack depicted in **Figure 2**.

The stack consists of the following layers:

- **Host:** The process that runs the application and can be anything from IIS or a standalone executable, to your own custom program. The host is responsible for startup, loading of other OWIN components and shutting down gracefully.
- **Server:** Responsible for binding to a TCP port, constructing the environment dictionary and processing requests through an OWIN pipeline.
- **Middleware:** The name given to all of the components that handle requests in an OWIN pipeline. It can range from a simple compression component to a complete framework such as ASP.NET Web API, though from the server's perspective, it's simply a component that exposes the application delegate.
- **Application:** This is your code. Because Katana is not a replacement for ASP.NET but rather a new way to compose and host components, existing ASP.NET Web API and SignalR applications remain unchanged, as those frameworks can participate in an OWIN pipeline. In fact, for these kinds of applications, Katana components will be visible only in a small configuration class.

Architecturally, Katana is factored such that each of these layers can be easily substituted, often without requiring a rebuild of the code.

Figure 2 **Katana Project Architecture**

When processing an HTTP request, the layers work together in a manner similar to the data flow shown in **Figure 3**.

Building a Modern Web Application with Katana

Modern Web applications generally have four capabilities:

1. Server-side markup generation
2. Static file serving
3. Web API for handling AJAX requests
4. Real-time messaging

Building an application with all of these capabilities requires a variety of different frameworks suitably specialized for the relevant capability. However, composing an application from such frameworks can be challenging, and currently requires hosting the different application parts on IIS, possibly isolating them from one another using applications and virtual directories.

In contrast, Katana lets you compose a modern Web application from a wide range of different Web technologies and then host that application wherever you wish, exposed under a single HTTP endpoint. This provides several benefits:

- Deployment is easy because it involves only a single application rather than one application per capability.
- You can add additional capabilities, such as authentication, which can apply to all of the downstream components in the pipeline.
- Different components, whether Microsoft or third-party, can operate on the same request state via the environment dictionary.

Now I'll explore a sample application that has a domain you should be familiar with: bug tracking. The application will present a set of bugs in a variety of different states—backlog, working and done—and allow me to move the bug between states. And, because many different individuals could be managing bugs simultaneously, it will update all browsers in real time when the state of a bug changes. Here's what I'll use to build the application: Nancy (nancyfx.org) for server-side markup generation and static file serving; ASP.NET Web API (asp.net/web-api) for handling AJAX requests; and SignalR (signalr.net) for real-time messaging services.

Additionally, while I'm not going to spend a great deal of time on markup and script for the browser client, I'll use Knockout.js to separate the HTML markup from the Web API and SignalR data.

The core principle to keep in mind is that I'm composing all of these different frameworks into a single OWIN pipeline, so as new capabilities become available I can add them to the application by simply inserting them into the pipeline.

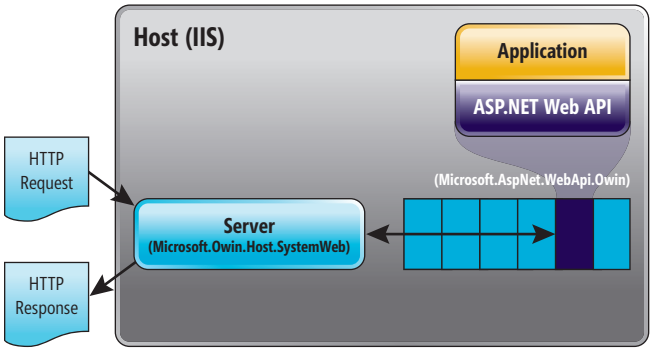


Figure 3 **Example of the Data Flow in Katana**

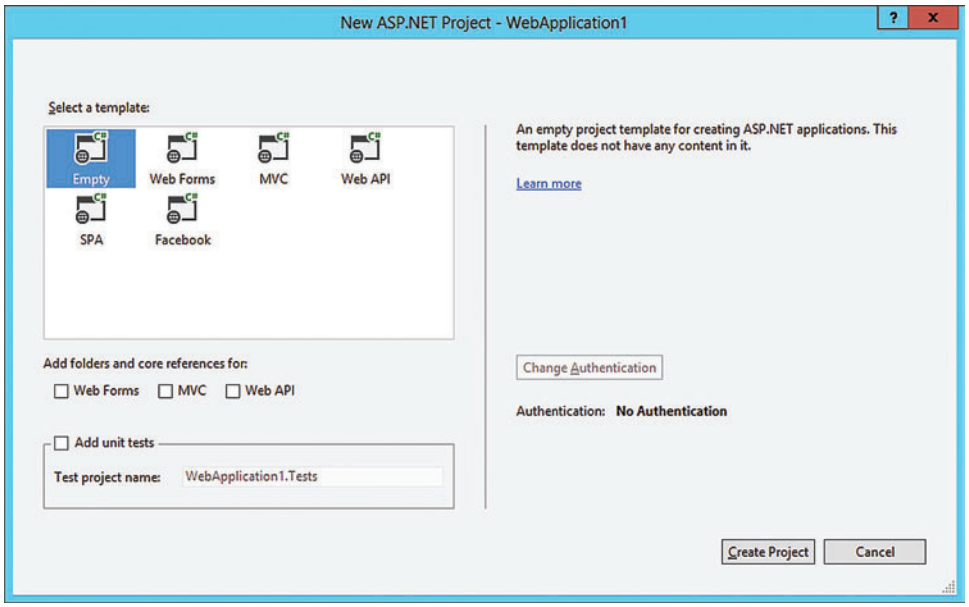


Figure 4 **A New ASP.NET Web Application Project in Visual Studio 2013 Preview**

Getting Started

One of the goals of Katana is to give you precise control over the capabilities added to your app (and, therefore, over what you spend in terms of performance for processing each request). With that in mind, I'll begin by creating a new Empty ASP.NET Web app project in Visual Studio 2013 Preview, as shown in **Figure 4**.

Web project templates, even empty ones, provide a helpful feature in that, by default, they place compiled assemblies directly into the /bin folder rather than the /bin/debug folder (as is common in other project types). The default Katana host looks for assemblies in this /bin folder. You could create a Katana-based application as a class library, but you'd need to either modify your project properties to conform to this structure or supply your own custom application loader that could search for assemblies and types in a different folder structure.

Next, I'll build out the server-side markup generation code using the Nancy Web framework.

Nancy has a terse syntax, which makes it easy to quickly build HTTP-based sites and services. What's more important for this exercise is that, like the ASP.NET Web API, it doesn't have any dependencies on System.Web.dll and it's built to run in an OWIN pipeline. Frameworks such as ASP.NET MVC have dependencies on System.Web.dll (at the time of this writing), which make them less ideal for non-IIS hosting scenarios.

Figure 5 **Home.html**

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>@Model.title</title>
</head>
  <body>
    <header>
      <h1>@Model.title</h1>
    </header>
    <section>
      <h2>Backlog</h2>
      <ul class="bugs" id="backlog">
        <li>a bug</li>
      </ul>
    </section>
    <section>
      <h2>Working</h2>
      <ul class="bugs" id="working">
        <li>a bug</li>
      </ul>
    </section>
    <section>
      <h2>Done</h2>
      <ul class="bugs" id="done">
        <li>a bug</li>
      </ul>
    </section>
  </body>
</html>
```

For the most part, when you add new functionality to the application, you'll start by adding a NuGet package. (You can read more about NuGet at docs.nuget.org.) At the time of this writing, many of the packages being used here are prerelease versions, so make sure to allow prerelease packages to be displayed in the NuGet dialog.

To add Nancy to the application, I could simply install the Nancy NuGet package. However, because I also want to run Nancy in an OWIN pipeline, I'm going to install the Nancy.Owin package (nuget.org/packages/nancy.owin). This will install the Nancy package as a dependency and provide additional helper methods for configuring Nancy in my OWIN pipeline.

Next, I need to create a Nancy module (similar to a Model-View-Controller, or MVC, controller) to handle requests, as well a view to display something to the browser. Here's the code for the module (HomeModule.cs):

```
public class HomeModule : NancyModule
{
  public HomeModule() {
    Get["/"] = _ => {
      var model = new { title = "We've Got Issues..." };
      return View["home", model];
    };
  }
}
```

As you can see, the module declares that requests directed to the application root ("/") should be handled by the anonymous delegate defined in the associated lambda. That function creates a model containing the page title and instructs Nancy to render the "home" view, passing the model to the view. The view, shown in **Figure 5**, inserts the model's title property into both the page title and h1 elements.

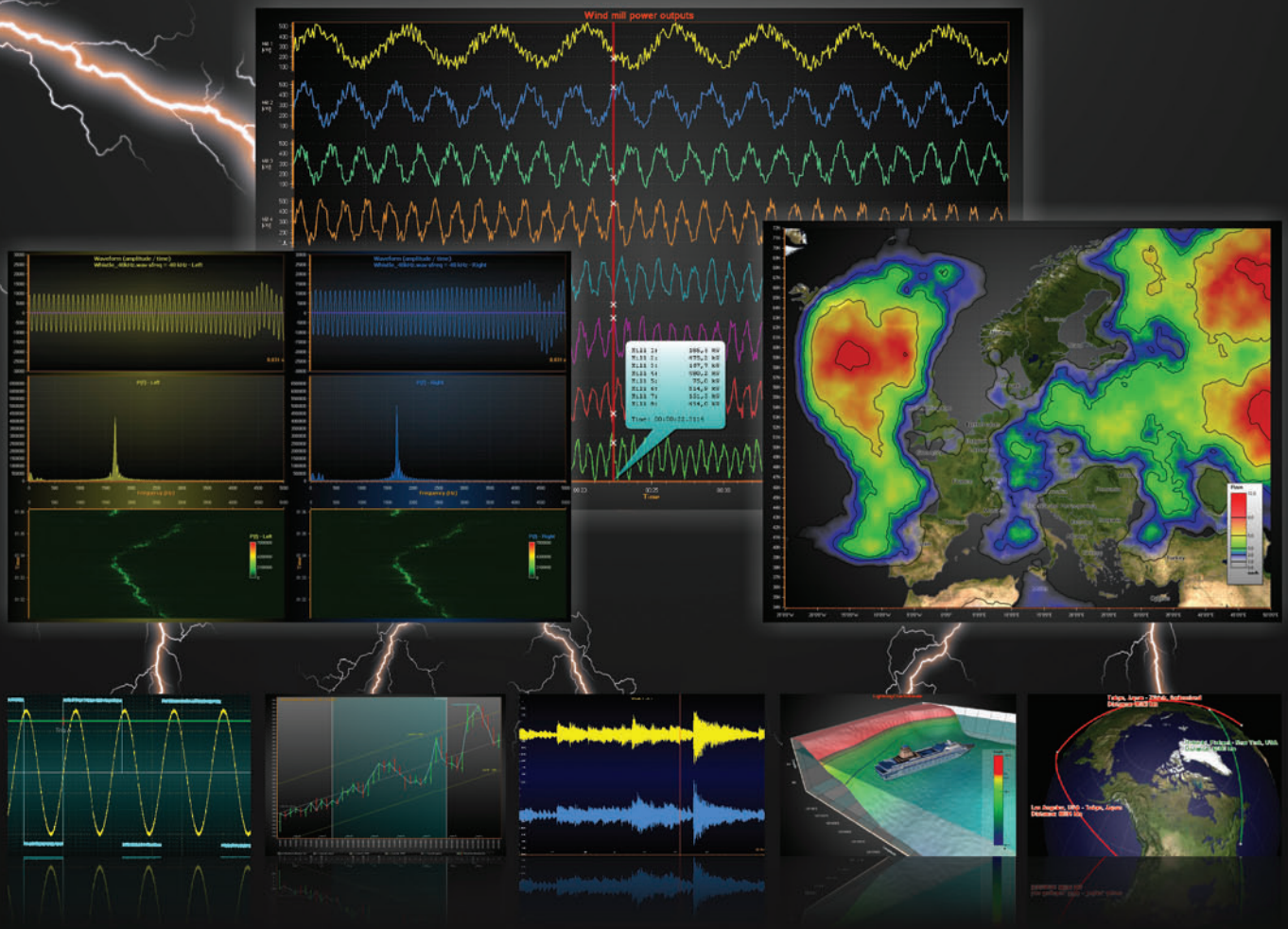
For more information about these listings, please refer to the Nancy documentation.

Now that I've implemented the basic Nancy functionality, I need to establish an OWIN pipeline and configure the Nancy module to participate in that pipeline. For this, I need to first install the Katana host and server components, then write a small amount of plumbing code to set up the OWIN pipeline and insert Nancy into that pipeline.

ULTIMATE CHARTING POWER

LightningChart

The fastest rendering data visualization components for WPF and WinForms



- **Fully DirectX accelerated**
- **Superior 2D and 3D rendering performance**
- **Very extensive property sets**
- **Optimized for real-time data monitoring**
- **Supports gigantic data sets**
- **Professional, friendly and fast customer support**
- **Compatible with Visual Studio 2005...2012**

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.

FREE TRIAL

Download a free 30-day evaluation from

www.LightningChart.com

Arction

Pioneers of high-performance data visualization

For the Katana host and server components, I'll begin by using IIS Express and System.Web, as these are inherently understood by Visual Studio and will enable a smooth F5 experience while building the application. To incorporate the System.Web host into the project, I install the NuGet package Microsoft.Owin.Host.SystemWeb (bit.ly/19EZ2Rw).

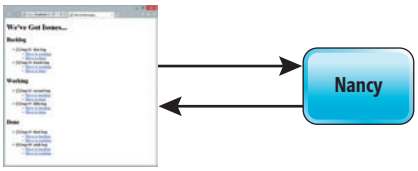


Figure 6 **A Functional Web Application with a Single Component**

The default Katana components use several different conventions for loading and running OWIN applications, including the startup class. When a Katana host loads an OWIN application, it discovers and runs a startup class based on the following rules (in order of precedence):

- If the web.config file contains an appSetting with key="owin:AppStartup", the loader uses the setting value. The value must be a valid .NET-type name.
- If the assembly contains the attribute [assembly: OwinStartup(typeof(MyStartup))], the loader will use the type specified in the attribute value.
- If neither of these conditions are true, the loader will scan the loaded assemblies looking for a type named Startup with a method that matches the signature void Configure(IAppBuilder app).

For this example, I'll allow the loader to scan assemblies for the class. However, if you have many different types and assemblies in your project, it would be wise to use either the appSetting or assembly attribute to prevent unnecessary scanning.

I'll create the startup class that will initialize my OWIN pipeline and add Nancy as a pipeline component. I create a new class called Startup and add a configuration method as follows:

```
public class Startup
{
  public void Configuration(IAppBuilder app)
  {
    app.UseNancy();
  }
}
```

UseNancy is an extension method made available by the Nancy.Owin NuGet package. While you can add middleware using the IAppBuilder's more generic Use methods, many middleware libraries will provide these helpful extension methods that ease the configuration process.

At this point, you can run the project in Visual Studio using F5 and see that though it's not terribly exciting yet, you have a fully functional Web application. At this point, the OWIN pipeline consists of a single component, Nancy, as shown in **Figure 6**.

Incorporating Data with ASP.NET Web API

Currently, the HTML view consists of primary static markup. I'll now give users some real bugs with which to work. For many modern Web apps, the task of delivering data to the browser client has shifted from the server-side markup generation framework (like the Nancy module) to a separate Web API service. The browser, then, loads the HTML page and immediately executes JavaScript, which fetches the data from the Web API and dynamically builds HTML markup in the page itself.

I'll start by constructing the Web API using the ASP.NET Web API framework. As usual, the first step is to install the Web API NuGet package. To ensure I can easily insert ASP.NET Web API into my OWIN pipeline, I'll install the Microsoft.AspNet.WebApi.Owin package (bit.ly/1dnocmK). This package will install the rest of the ASP.NET Web API framework as dependencies. After installing the framework, I'll create a simple API as shown in **Figure 7**.

The API contains a method to return a set of bug objects from a repository, as well as some methods to move bugs between different states. Much more information about the ASP.NET Web API can be found at asp.net/web-api.

Now that I have an ASP.NET Web API controller defined, I need to add it to my existing OWIN pipeline. To do this, I simply add the following lines to the Configuration method in my startup class:

```
var config = new HttpConfiguration();
config.MapHttpAttributeRoutes();
config.Routes.MapHttpRoute("bugs", "api/{Controller}");

app.UseWebApi(config);
```

Just like Nancy, the ASP.NET Web API OWIN package provides the UseWebApi extension method, making it easy to incorporate the ASP.NET Web API into my existing OWIN pipeline. The OWIN pipeline now consists of two components, ASP.NET Web API and Nancy, as shown in **Figure 8**.

As requests come into the pipeline, if they match one of the ASP.NET Web API routing rules, the ASP.NET Web API handles the request and generates a response. Otherwise, the request continues through the pipeline, where Nancy attempts to handle it. If no pipeline components can handle a particular request, the default Katana components will return an HTTP 404 response.

Figure 7 **BugsController.cs**

```
public class BugsController : ApiController
{
  IBugsRepository _bugsRepository = new BugsRepository();

  public IEnumerable<Bug> Get()
  {
    return _bugsRepository.GetBugs();
  }

  [HttpPost("api/bugs/backlog")]
  public Bug MoveToBacklog([FromBody] int id)
  {
    var bug = _bugsRepository.GetBugs().First(b=>b.id==id);
    bug.state = "backlog";
    return bug;
  }

  [HttpPost("api/bugs/working")]
  public Bug MoveToWorking([FromBody] int id)
  {
    var bug = _bugsRepository.GetBugs().First(b => b.id == id);
    bug.state = "working";
    return bug;
  }

  [HttpPost("api/bugs/done")]
  public Bug MoveToDone([FromBody] int id)
  {
    var bug = _bugsRepository.GetBugs().First(b => b.id == id);
    bug.state = "done";
    return bug;
  }
}
```

Although I have a functioning ASP.NET Web API, it's currently not being accessed by the home view. Therefore, I'll add code to consume data from the Web API and generate a list of bugs in each of the different states: backlog, working and done. For this task, I'll take advantage of Knockout.js, a JavaScript Model-View-ViewModel (MVVM) library. More information about Knockout can be found at knockoutjs.com.

To enable dynamic, client-side generation of HTML markup using Knockout, the first thing I need to do is fetch all bugs from the ASP.NET Web API and create a viewModel that Knockout can bind to HTML elements. This is shown in **Figure 9**.

Once the viewModel is created, Knockout can then dynamically generate and update HTML content by binding the viewModel to HTML elements decorated with Knockout-specific attributes. For example, the backlog list can be generated from the viewModel using the attributes shown in **Figure 10**.

Adding Real-Time Change Notifications with SignalR

At this point, I have a fully functioning single-page Web application. Users can browse to the home view and move bugs between different bug states. Moreover, the underlying technologies for the current level of functionality, Nancy and ASP.NET Web API, are running together in the same OWIN pipeline.

I'm going to go one step further, however, and allow different users to see, in real time, the updates made to bugs by other users. For this I'll leverage the SignalR library, which provides both a client and server API for managing real-time message exchanges between the browser and a Web server. SignalR is also written to run in an OWIN pipeline, so adding it to my existing application will be trivial.

Figure 9 **Setting up the Bugs viewModel**

```
<script>
  $(function () {
    var viewModel;

    $.getJSON('/api/bugs', function(data) {
      var model = data;
      viewModel = {
        backlog: ko.observableArray(
          model.filter(function(element) { return element.state === 'backlog'; })),
        working: ko.observableArray(
          model.filter(function(element) { return element.state === 'working'; })),
        done: ko.observableArray(
          model.filter(function(element) { return element.state === 'done'; })),
        changeState: function (bug, newState) {
          var self = this;
          $.post('/api/bugs/' + newState, { '': bug.id }, function(data){
            self.moveBug(data);
          });
        },
        moveBug: function (bug) {
          // Remove the item from one of the existing lists
          ...

          // Add bug to correct list
          this[bug.state].push(bug);
        }
      };
      ko.applyBindings(viewModel);
    })
  })
</script>
```

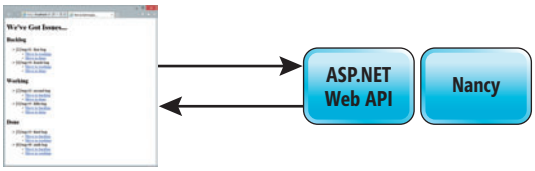


Figure 8 **The OWIN Pipeline with Two Components**

I'll use a SignalR feature called Hubs, and while the details of SignalR are beyond the scope of this article, a Hub enables clients and servers to call methods on one another. (For a great intro to SignalR, see bit.ly/14Wlx1t.) In my application, when the ASP.NET Web API receives a request to change the state of a bug, it will update the bug and then broadcast the updated bug through the SignalR Hub to all browser clients currently connected to the application.

I'll start this process by creating a Hub on the server. Because I'm not taking advantage of any additional SignalR capabilities, my Hub will consist simply of the following empty class definition:

```
[HubName("bugs")]
public class BugHub : Hub
{
}
```

In order to send broadcasts to the Hub from the ASP.NET Web API, I first need to get an instance to its runtime context. I can do this by adding the following BugsController constructor:

```
public BugsController()
{
  _hub = GlobalHost.ConnectionManager.GetHubContext<BugHub>();
}
```

From within one of the MoveToXX actions, I can then broadcast the updated bug to all of the connected browser clients:

```
_hub.Clients.All.moved(bug);
```

In the home view, after adding a couple of script references to the SignalR JavaScript libraries, I can connect to the bugsHub and start listening for "moved" messages with the following:

```
$.connection.hub.logging = true;
var bugsHub = $.connection.bugs;

bugsHub.client.moved = function (item) {
  viewModel.moveBug(item);
};

$.connection.hub.start().done(function() {
  console.log('hub connection open');
});
```

Notice that when I receive a call from the server via the moved function, I call the viewModel moveBug method in the same way I did in the item's click handler. The difference is that because this method is the result of a SignalR broadcast, all browser clients can update their viewModels at the same time. You can see this clearly by opening two browser windows, making changes in one, and then viewing the state change in the other.

Figure 10 **Attributes for Generating the Backlog List**

```
<section>
  <h2>Backlog</h2>
  <ul class="bugs" id="backlog" data-bind="foreach:backlog">
    <li>
      [<span data-bind="text: id"></span>] <span data-bind="text: title"></span>:
        <span data-bind="text: description"></span>
      <ul>
        <li><a href="#" data-bind="click: $root.changeState.bind($root, $data,
          'working')">Move to working</a></li>
        <li><a href="#" data-bind="click: $root.changeState.bind($root, $data,
          'done')">Move to done</a></li>
      </ul>
    </li>
  </ul>
</section>
```

ComponentOne Studio Enterprise 2013 v2 | from $1,315.60

ComponentOne
a division of GrapeCity

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

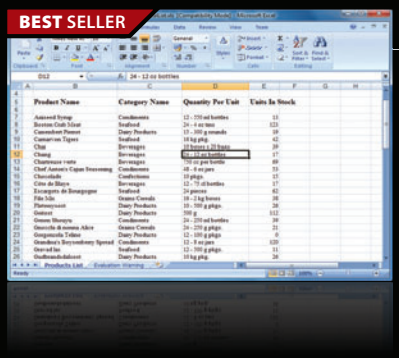
- Hundreds of UI controls for all .NET platforms including grids, charts, reports and schedulers
- New RadialMenu, RichTextBox and OrgChart for Windows Store Apps
- Ready -to-use Mobile Project Templates and custom Mobile Scaffolding
- Touch support in WPF and Silverlight controls
- Royalty-free deployment and distribution

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



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

ASPOSE
Your File Format Experts

Every Aspose .NET component in one package.

- Programmatically manage popular file formats including Word, Excel, PowerPoint and PDF
- Work with charts, diagrams, images, Project plans, emails, barcodes, OCR, and document management 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

GdPicture.NET | from $4,600.56

GdPicture
imaging technologies

All-in-one AnyCPU document-imaging and PDF toolkit for .NET and ActiveX.

- Document viewing, processing, printing, scanning, OMR, OCR, Barcode Recognition
- Annotate image and PDF within your Windows & Web applications
- Read, write and convert vector & raster images in more than 90 formats, including PDF
- Color detection engine for image and PDF compression
- 100% royalty-free and world leading Imaging SDK

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

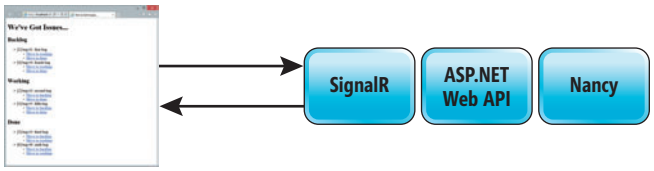


Figure 11 **The OWIN Pipeline with Three Components**

As I noted, adding SignalR to the OWIN pipeline is trivial. I simply add the following to the startup class Configuration method:

```
app.MapSignalR();
```

This creates a pipeline like the one in **Figure 11**.

Moving to Self-Host

I now have a functioning bug-management application that, while still missing some key features, can do a few interesting things. I've incrementally added capabilities to the application by using both Microsoft and third-party Katana middleware components. However, much of this was already possible today using ASP.NET HttpModules and HttpHandlers. So what have I really accomplished, other than providing a simpler, code-driven approach to compose the pipeline components?

The key is to remember the high-level Katana architecture diagram in **Figure 2**. So far, I've just been working at the top two layers of the Katana stack. However, all of these layers can be easily replaced, including the server and host.

To demonstrate, I'll take my entire pipeline, lift it out of IIS and System.Web.dll, and sit it on top of a simple, lightweight HTTP server, which is being hosted by a Katana executable named OwinHost.exe. Self-hosting can prove useful in a variety of scenarios, ranging from setups where there's no Web server installed on the development machine, to production situations where the application is being deployed in a shared-hosting environment that uses process isolation and doesn't expose access to a Web server.

I'll start by installing the following additional NuGet packages:

- Microsoft.Owin.Host.HttpListener (bit.ly/153alca)
- OwinHost (bit.ly/162Uzj8)

I'll then rebuild the application. Note that rebuilding is not required in order to run the application on top of a new server and host. The only requirement is that those files exist in the /bin folder at run time, and rebuilding is a convenient way to have the files copied into /bin.

After the packages have been installed and the files copied, I open a command prompt, navigate to the Web project's root folder, and, as shown in **Figure 12**, call OwinHost.exe from within the packages folder:

```
> ..\packages\OwinHost.2.0.0\tools\OwinHost.exe
```

By default, OwinHost.exe will launch, load the Microsoft.Owin.Host.HttpListener server, and begin listening on port 5000. I can then navigate to http://localhost:5000 to confirm the entire application is running.

Furthermore, nearly all of the defaults can be overridden using command-line switches. For example, if you want to listen on a different port, supply -p 12345. If you want to use a completely different server, use -s your.custom.server.assembly. The power of the Katana design is its modularity. As innovations happen at any layer of the stack, they can be immediately integrated into running applications. And because the contract between all components of the stack is simply the application delegate, the pace of innovation can be much greater than what's available now.

Just Getting Started

Katana 2.0 will be released with Visual Studio 2013. The new release has two main areas of focus:

- Providing the core infrastructure components for self-hosting
- Providing a rich set of middleware for authentication, including social providers such as Facebook, Google, Twitter, and Microsoft Account, as well as providers for Windows Azure Active Directory, cookies, and federation

Once Katana 2.0 is released, work will begin immediately on the next set of Katana components. The details and priorities are still being determined, but you can influence that discussion by filing issues at katanaproject.codeplex.com. Finally, all of the code for this article can be found at bit.ly/1alOF4m. ■

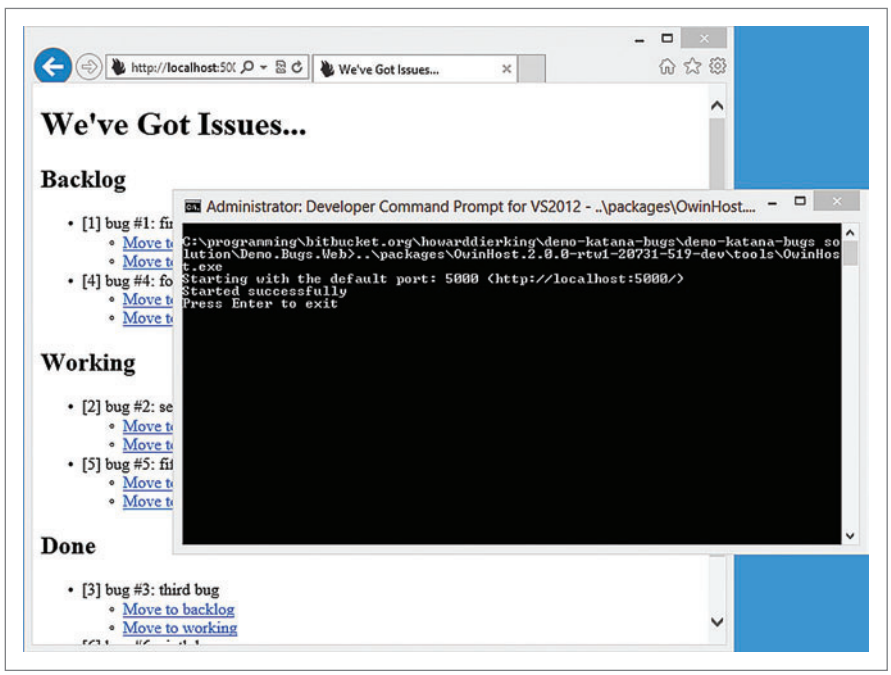


Figure 12 **Calling OwinHost.exe from Within the Packages Folder**

Howard Dierking *is a program manager on the Windows Azure Frameworks and Tools team where his focus is on ASP.NET, NuGet and Web APIs. Previously, Dierking served as the editor in chief of* MSDN Magazine, *and also ran the developer certification program for Microsoft Learning. He spent 10 years prior to Microsoft as a developer and application architect with a focus on distributed systems.*

Thanks to the following Microsoft technical experts for reviewing this article:
Brady Gaster and Scott Hanselman

Building an Alarm App in Windows 8.1

Tony Champion

Among the hundreds of new features in Windows 8.1 is the concept of an alarm app. In a nutshell, an alarm app is a Windows Store app that can schedule toast notifications to the second. Due to the way Windows processes toasts, this isn't an accuracy offered to most apps. In this article, I'll explore the concept of an alarm app and look at what it takes to develop one of your own.

What Is an Alarm App?

Before looking under the hood of an alarm app, it's important to consider what type of apps would make good alarm apps. The key is to consider the accuracy of delivering notifications. For example, it isn't necessary for a calendar app to notify the user of a friend's birthday at an exact second. However, there are several types of apps that do require this accuracy.

The most obvious choice is a good old-fashioned alarm clock app. When a user sets an alarm, your app needs to notify the user at that moment. You can now create more accurate time-management apps, such as a Pomodoro app, with an alarm app. An alarm app could also be used in short interval workouts, such as Tabata, where the timing of rounds and rests is extremely important.

All Windows Store app developers should be familiar with the Windows 8 concept of lock screen apps. Apps that are placed on the lock screen are allowed to post updates to the lock screen. In addition, a lock screen app has access to more capabilities than a typical app. This is important when your app needs to do things such as update the lock screen or run many of the available background tasks.

The need for the additional capabilities in lock screen apps is a direct result of changes in Windows 8 to improve performance and battery life. Windows is designed to prevent apps from running when they aren't being used and to consolidate tasks when possible. For this reason, you'll find scheduled events, such as background tasks and notifications, will always run at an approximate time. Windows batches these updates and runs them when it has a process ready to do the work.

In Windows 8 you could define up to seven apps to be on the lock screen at any given time. Included in these seven slots was a special slot for a detailed status app. A device might have seven

This article discusses:

- The basics of a new alarm app
- Setting up an alarm app
- Requesting alarm access
- Scheduling, managing and removing alarms
- Adding commands
- Working with snooze
- Changing the alarm sound

Technologies discussed:

Windows 8.1

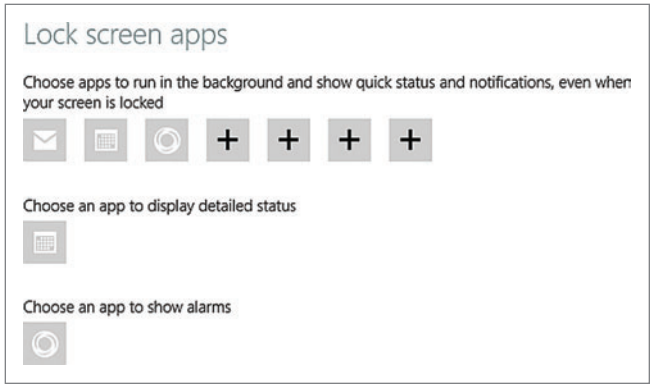


Figure 1 **Lock Screen Part of PC Settings**

lock screen apps that can display badges or text, but only one app that's able to provide a custom UI to the lock screen. Windows 8.1 keeps this setup and adds the alarm app as a special new type of lock screen app. **Figure 1** shows the new "Lock screen apps" settings part of the PC settings screen. Like the detailed status app, each device can only have a single alarm app at a time.

The sole difference between an alarm app and other lock screen apps is the ability to provide alarm toast notifications to the user within a second of the scheduled delivery time. As previously mentioned, submitting or scheduling a toast in a Windows Store app doesn't guarantee when that toast will be delivered. It's delivered in an approximate time frame, but the exact time is up to Windows. If an app is identified by the user as the alarm app, scheduled toasts are delivered exactly on time.

Setting Up an Alarm App

Before an app can be selected as an alarm app, the app manifest must be properly configured. If the manifest doesn't include the required features, then attempting to set the app as the alarm app in code will generate an error and the user won't have the option to set the app manually.

Because an alarm app schedules toasts, you first need to enable toasts. You can do this in the manifest designer in the Application UI tab. A toast-capable dropdown box can be found in the Notifications section under Visual Assets. The Notifications section can be found under All Image Assets as well as the Badge Logo subgroup. To enable toasts for the app, you should set the toast-capable dropdown to Yes.

The other property in the Notifications section is the ability to enable lock screen notifications for the app. As mentioned earlier, the alarm app is a special type of lock screen app; therefore the app must be configured to be on the lock screen. You should set the lock screen notifications dropdown to either Badge or Badge and Tile Text.

Once you've enabled lock screen notifications on an app, the manifest needs to have a few additional items. For example, a badge logo must be assigned to the app. The logo is required to be a 24x24 pixel image, and like all logos in a Windows Store app, you can specify various-scale sizes to handle Windows resolution scaling. **Figure 2** shows an example of the manifest designer with toasts and lock screen notifications enabled.

Any app that has lock screen notifications enabled must also declare a background task in the app manifest. The interesting part of this requirement is that you don't need to actually implement a background task in a lock screen app; it just has to be declared.

A background task is declared in the Declarations section of the app manifest designer. In the Available Declarations, select Background Tasks and click Add. The background task for a lock screen app must support one of the following task types: Control channel, Timer, Push notification or Location. In the App settings section of the background task, you must set either the Entry point or Start page value. In a XAML app that's using the background task, the Entry point is set to the class that implements the IBackgroundTask interface. If the app isn't actually implementing a background task, the value can be set to anything. **Figure 3** shows a properly configured background task to enable an app to be added to the lock screen.

The final step in configuring the app manifest is to identify the app as an alarm app. You do this by adding an alarm extension in the manifest. Unfortunately, it isn't possible to add this extension through the manifest designer; it must be done by hand. In order to do that, you must first get to the underlying XML of the package manifest. Do this by right-clicking package.appxmanifest in the Solution Explorer and selecting View Code.

The XML in a Windows 8.1 app manifest will use two namespaces. The first namespace (which is the default) contains the elements defined in Windows 8. The second namespace must be added and is identified by the m2 prefix. It includes additions and changes added in Windows 8.1. Because the alarm app feature is new to Windows 8.1, you need to add an "8.1" extension to the Extensions collection within the Application element. The Category property of the new extension should be set to windows.alarm. Here's an example of what the Extensions collection should look like with the new extension and the previously declared background task:

```
<Extensions>
  <Extension Category="windows.backgroundTasks" EntryPoint="App">
    <BackgroundTasks>
      <Task Type="timer" />
    </BackgroundTasks>
  </Extension>
  <m2:Extension Category="windows.alarm" />
</Extensions>
```

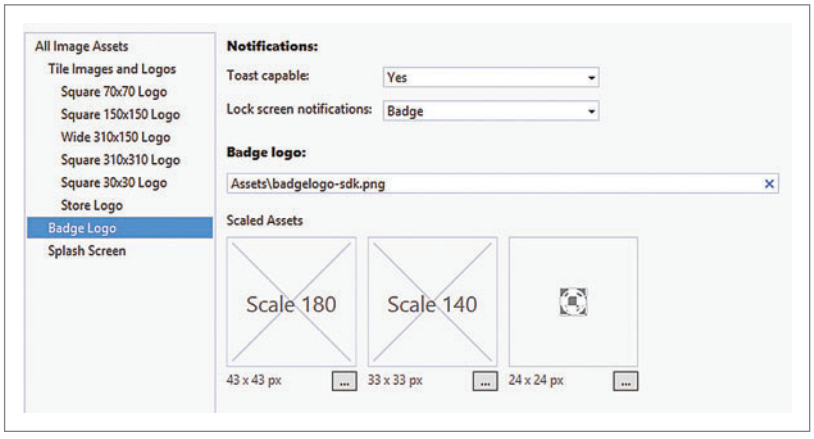


Figure 2 **Notifications Section of the Manifest Designer**

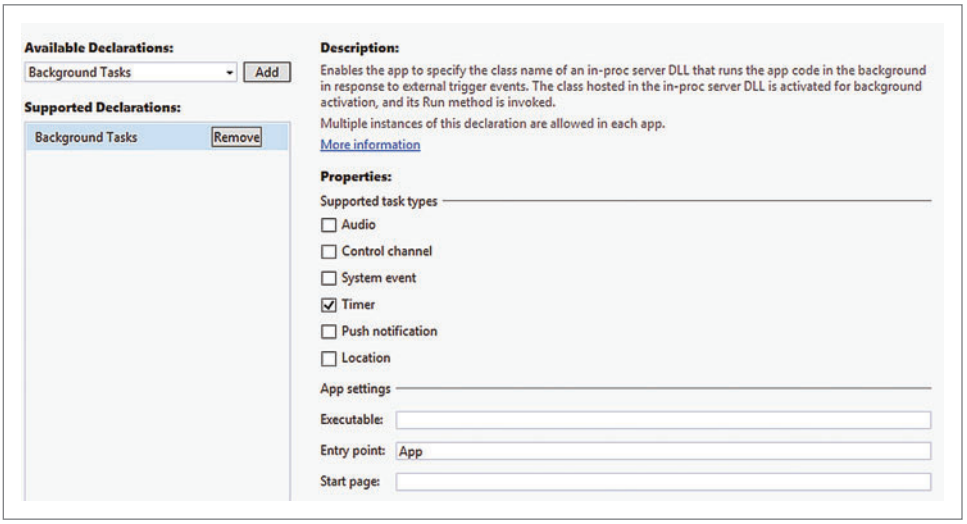


Figure 3 **Configuring a Background Task**

Requesting Alarm Access

Once the appropriate settings and declarations have been made, your app can then request access to be set as the alarm app for the device. The user can do this manually from the PC Settings; however, your app can request access through the Windows Runtime (WinRT).

The Windows Runtime includes a static AlarmApplicationManager class in the Windows.ApplicationModel.Background namespace. This class has a RequestAccessAsync method that will prompt the user for permission to set the app as the device alarm app, as shown in **Figure 4**. If a different app is currently identified as the alarm app, it will be replaced by the current one if the user selects Yes when prompted.

The RequestAccessAsync method returns an AlarmAccessStatus enum that has three valid values: Denied, AllowedWithWakeupCapability and AllowedWithoutWakeupCapability. This is important if you're creating alarm apps for things such as morning wakeup alarms. If the device is in sleep mode and it isn't configured to allow toast notifications to wake it, then your alarms won't fire.

It's important to remember that Windows 8 puts the user in control of a device. As shown in **Figure 1**, a user can easily change the alarm app. Your challenge is that each app can only request access from the AlarmApplicationManager once. If the app makes a second request for access—for this user and device—it won't prompt the user and will only return the current alarm status of the app. Once your app has been denied, replaced by a different app or manually removed by the user, your only option is to inform the user that the app needs to be manually added back as the alarm app for the device.

The AlarmApplicationManager also includes a GetAccessStatus method that returns the current alarm status of the machine. As with the RequestAccessAsync method, it returns an AlarmAccessStatus enum and is a great way to determine what's going on with the device. For example, your app could inform the user if notifications won't wake up the device.

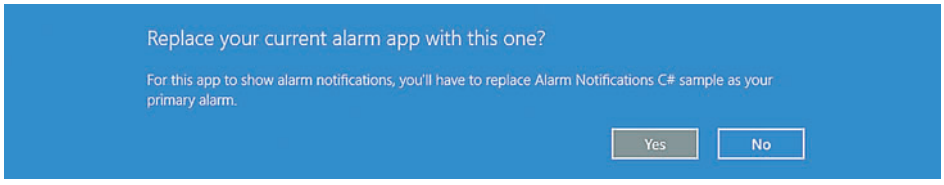


Figure 4 **An Alarm App Permission Prompt**

Scheduling Alarms

Alarms are actually just a scheduled toast notification with a little more kick. The process of scheduling an alarm is identical to scheduling a toast notification, the only difference being the notification XML. The XML used to define a notification will contain information about its appearance as well as any included features.

A toast can be either just text or a combination of text and an image, as defined by one of the templates provided by the Windows Runtime. There are currently eight different XML templates from which you can choose. As of this writing, Windows 8.1 doesn't support custom toast layouts, so you must select from one of the provided templates.

The type and layout of a toast is defined in a block of XML. The XML identifies the toast template to use, the text and image with which to populate the template, and several other options that I'll discuss later. Here's an example of a basic toast notification:

```
<toast duration="long">
  <visual>
    <binding template="ToastText02">
      <text id="1">Sample Toast App</text>
      <text id="2">The is a sample message.</text>
    </binding>
  </visual>
</toast>
```

There are several ways to generate the toast template. The Windows Runtime will create an XmlDocument of each template from the ToastNotificationManager, which can be found in the Windows.UI.Notification namespace. The ToastNotificationManager has a static GetTemplateContent method that takes a ToastTemplateType enum that has a value for each of the available templates. Here's an example of how to get an XmlDocument of the previous example:

```
XmlDocument content =
  ToastNotificationManager.GetTemplateContent(ToastTemplateType.ToastText02);
content.DocumentElement.SetAttribute("duration", "long");
var textLines = content.GetElementsByTagName("text");
textLines[0].InnerText = "Sample Toast App";
textLines[1].InnerText = "The is a sample message.";
```

This is a great starting point; however, depending on your usage, it can be a bit cumbersome. This is why any additions or modifications to the template will normally be done using the XmlDocument API. This can lead to quite a few lines of code for a few simple additions. With this in mind, it isn't uncommon to find the XML built up as a string and then loaded into an

Figure 5 **Generating XML As a String**

```
string textLine1 = "Sample Toast App";
string textLine2 = "This is a sample message.";

string contentString =
  "<toast duration=\"long\">\n" +
    "<visual>\n" +
      "<binding template=\"ToastText02\">\n" +
        "<text id=\"1\">" + textLine1 + "</text>\n" +
        "<text id=\"2\">" + textLine2 + "</text>\n" +
      "</binding>\n" +
    "</visual>\n" +
  "</toast>\n";

XmlDocument content = new Windows.Data.Xml.Dom.XmlDocument();
content.LoadXml(contentString);
```

XmlDocument once the template is complete. **Figure 5** shows how to achieve the equivalent of the previous example by generating the XML as a string.

If you're going this route, you should be careful of two things. First, you must make sure you're generating properly formatted XML—take care to escape any special characters that might be included in the string. Second, make sure you follow the schema of the toast notification. The Windows Dev Center provides the full specifications for the schema at bit.ly/172oYCO.

All of the previous examples have one element that isn't part of the default template. You might have noticed the duration attribute on the toast element has been set to long. Because you're creating alarm toast notifications, you don't want the alarm to appear and then vanish without user interaction. Setting the duration to long will keep the toast on the screen for up to 25 seconds.

Once you have your XML configured and loaded into an XmlDocument, you can schedule your toast. The first step is to create an instance of a ScheduledToastNotification. You need two things to create a ScheduledToastNotification: the XmlDocument containing the toast definition and a time for the toast to be scheduled.

The ScheduledToastNotification is scheduled with a ToastNotifier. An instance of a ToastNotifier is created from the ToastNotificationManager class using the static CreateToastNotifier method. A call to the ToastNotifier AddToSchedule method will then schedule your toast to be delivered. Toasts typically are created as the result of some user action, such as pressing a button, or an event that's raised within the app. Here's an example of creating and scheduling a toast for 1 minute from the current time:

```
ToastNotifier toastNotifier =
  ToastNotificationManager.CreateToastNotifier();
var scheduledToast = new ScheduledToastNotification(
  content, DateTime.Now.AddMinutes(1));
toastNotifier.AddToSchedule(scheduledToast);
```

The resulting toast that's generated can be seen in **Figure 6**. Notice it should look like most toasts you've seen in Windows.

A toast notification might fail for a few different reasons. The most frequent reason is that the XML is improperly formatted. This can be caused by having an incorrect template id or not defining all of the required attributes, so it's important to make sure you get the XML right.

If you attempt to schedule a toast for a time that has already passed, an exception will be thrown during the AddToSchedule call. If you're scheduling a toast for a few seconds in the future, make sure the time calculation is one of the last things you do. Setting a time for 2 seconds from now and then taking 4 seconds to execute code before actually scheduling that toast will throw an exception.

Finally, your app can only have 4,096 toasts scheduled at a time. I know your first thought is, "no way that will happen," but for an alarm app it's easier to do than you might think. Suppose you have an alarm app that allows the user to schedule a daily alarm. If your code schedules multiple daily alarms for each day for the next year, then 12 daily alarms will max you out. This means you need to make sure your algorithms for scheduling alarms are thought out and that you manage the alarms once they're created.

Managing and Removing Alarms

To successfully build an alarm app, you need to have the ability to review and remove any scheduled alarms. The ToastNotifier can accomplish both of these tasks. Calling the GetScheduledToastNotifications method will return a read-only list of ScheduledToastNotifications. The collection will only contain scheduled notifications for the current app.

To remove a scheduled notification, the notification to be removed should be passed to the RemoveFromSchedule method. Between these two methods, you can maintain your scheduled notifications. Once a notification has been posted and then dismissed by the user, it will no longer appear in the app's collection of notifications and won't count toward the maximum number of allowed scheduled notifications. Here's how to clear out all of the scheduled notifications:

```
var toastNotifier = ToastNotificationManager.CreateToastNotifier();
var notifications = toastNotifier.GetScheduledToastNotifications();

// Remove each notification from the schedule
foreach (var notification in notifications)
{
  toastNotifier.RemoveFromSchedule(notification);
}
```

You could put this code in a command or event handler tied to a Clear All button, for example.

Adding Commands

If your app is currently the alarm app for the device, then the previous example that generated the toast shown in **Figure 6** will display on time, but **Figure 6** doesn't look much like an alarm, does it? Primarily, you're missing a couple of common features for an alarm: snooze and dismiss. By default, clicking on the toast or swiping it will dismiss it; however, an alarm needs to be a bit more intuitive.

In addition to the <toast> element supporting a <visual> child element, it also supports a <commands> element. This element allows you to add predefined commands to the toast. The element

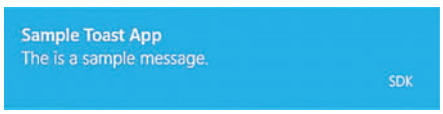


Figure 6 **A Basic Toast Notification**

Figure 7 **An Alarm Toast Notification**

itself supports a single optional scenario attribute. The scenario attribute can be set to either alarm or incomingCall. For my purpose here, it will always be set to alarm.

The <commands> element contains a collection of individual <command> elements. Each command must have an id attribute identifying what type of command it is. For an alarm, this must be either snooze or dismiss. By adding a commands section to your previous toast, your toast now looks a bit more like an alarm and has the ability to not only be dismissed but to snooze as well. Here's the newly generated toast code (you can see the results of this code in **Figure 7**):

```
<toast duration="long">
  <visual>
    <binding template="ToastText02">
      <text id="1">Sample Toast App</text>
      <text id="2">The is a sample message.</text>
    </binding>
  </visual>
  <commands scenario="alarm">
    <command id="snooze"/>
    <command id="dismiss"/>
  </commands>
</toast>
```

Setting the Snooze

By default, Windows sets the snooze time for an alarm at 9 minutes. However, you can define the duration of the snooze in the ScheduledToastNotification class. This class has a second constructor with two additional parameters. The first is the length of time for the snooze. The snooze interval is defined as a TimeSpan and can range anywhere from 1 to 60 minutes.

The second new parameter is the maximum number of times the user can hit the snooze on the alarm. Setting this value to zero will allow the user to hit the snooze an unlimited amount of times. Here's an example of setting the snooze frequency:

```
DateTime scheduledTime = DateTime.Now.AddMinutes(1);
TimeSpan snoozeInterval = TimeSpan.FromMinutes(5);

var scheduledToast = new ScheduledToastNotification(
  content, scheduledTime, snoozeInterval, 0);
```

Changing the Alarm Sound

If the user has enabled sound for notifications, then each toast will make the same "ding" sound when a toast message is displayed. This causes a few problems when it comes to creating an alarm app. First, the user gets used to hearing the same sound for all notifications. This means an alarm can fade into the noise of the other notifications and not stand out. The second issue is that usually an alarm continues to make sound until the user acknowledges it. Therefore, you want an alarm to have some type of looping sound.

Fortunately, the toast definition also supports an <audio> element that allows you to customize the sound made when the toast appears. The <audio> element has two attributes: src and loop. Before you get too excited, the value of the src attribute must be one of the predefined values supplied by Windows. This means you can't provide custom audio for your toast notifications, but you do have a decent collection of sounds from which to choose.

The value of src must be one of 25 predefined values. The sounds are broken up into looping and non-looping categories. The looping sounds are constructed to seamlessly loop and should be used when creating a looping audio notification. There are 10 alarm looping sounds and 10 incoming call looping sounds provided. A complete list of the predefined values can be found at bit.ly/16HV2xm.

Figure 8 **Setting a Custom Looping Audio Sound for a Toast**

```
<toast duration="long">
  <visual>
    <binding template="ToastText02">
      <text id="1">Sample Toast App</text>
      <text id="2">The is a sample message.</text>
    </binding>
  </visual>
  <commands scenario="alarm">
    <command id="snooze"/>
    <command id="dismiss"/>
  </commands>
  <audio src="ms-winsoundevent:Notification.Looping.Alarm2" loop="true" />
</toast>
```

Figure 9 **Setting the Silent Attribute So a Toast Doesn't Play Any Sound**

```
<toast duration="long">
  <visual>
    <binding template="ToastText02">
      <text id="1">Sample Toast App</text>
      <text id="2">The is a sample message.</text>
    </binding>
  </visual>
  <commands scenario="alarm">
    <command id="snooze"/>
    <command id="dismiss"/>
  </commands>
  <audio silent="true" />
</toast>
```

If you want to play a looping sound, the loop attribute must be set to true. In addition, the duration attribute on the <toast> element must be set to long in order to give the sound time to play. **Figure 8** shows an example of setting a custom looping audio sound for your toast.

If you don't want your toast to play any sound, you can set a silent attribute to true. This will override any default settings on displaying toasts. **Figure 9** shows how it's used.

Thinking Beyond Your App

You can take an alarm app in many different directions. Because each device can only have a single alarm app, it's important to think beyond your app. If the scope of your app is too limited, then the user might replace it with a different app. This, of course, depends on the purpose for your app and its target audience. Consider things such as using the Share Target contract to allow other apps to schedule alarms through your app.

With the addition of the alarm app feature in Windows 8.1, you can now create a wide range of time-based apps with the accuracy expected by the user. I've shown what it takes to get you up and running with an alarm app. Now, no more hitting the snooze button. Go out and create the next amazing alarm app! ■

Tony Champion *is president of Champion DS and is a Microsoft MVP. He's active in the community as a speaker, blogger and author. He maintains a blog at tonychampion.net and can be reached via e-mail at tony@tonychampion.net.*

Thanks to the following technical expert for reviewing this article: *Pete Brown (Microsoft)*

NEVRON
visualize your success

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

Nevron Data Visualization

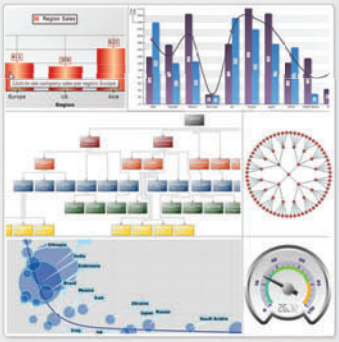
The leading data visualization components for a wide range of .NET platforms.14+ years of refinement, complete feature sets, highly customizable design and great support.

Developers

NEVRON Open VISION

Nevron Open Visions is a Cross - Platform Presentation Layer based on .NET. It aims to run on all major operating systems and integrate with already existing .NET based presentation layers.

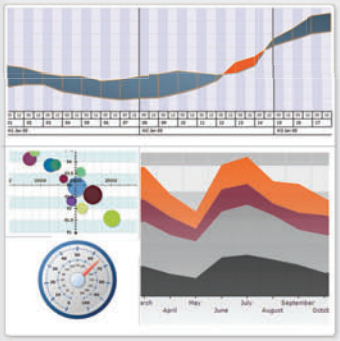
NEVRON VISION for .NET



Nevron Vision for .NET includes chart, gauge, diagram, map and user interfaces components that help you create enterprise-grade digital dashboards, scorecards, diagrams, maps, MMI interfaces and much more.

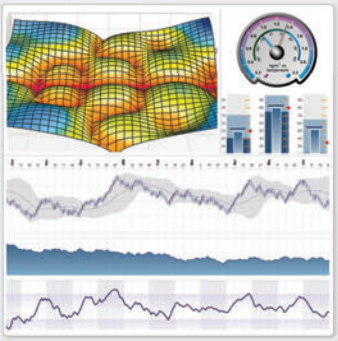
IT Professionals

NEVRON VISION for SharePoint



Nevron Vision for SharePoint combines the leading Chart and Gauge web parts for SharePoint 2007, 2010 and 2013. With this suite you can easily convert your SharePoint pages into interactive dashboards and reports.

NEVRON VISION for SSRS



Nevron Vision for SSRS presents the leading data visualization report items for SSRS 2005, 2008 and 2012. The Chart and Gauge help you deliver deep data insights with highly customizable engaging looks.

Nevron components integrate seamlessly in Web and Desktop .NET applications, SQL Server Reporting Services reports and SharePoint portals. All data visualization tools deliver an unmatched set of enterprise- grade features which makes Nevron the trusted vendor for many Fortune 500 companies.

Download you free evaluation copy from **www.nevron.com** today.

Programming the Nokia Sketch Effect in Windows Phone 8

Srikar Doddi

Nokia recently released the Nokia Imaging SDK in beta to allow Windows Phone 8 developers to create advanced imaging experiences for Nokia Lumia smartphones.

The Nokia Imaging SDK includes a library for manipulating images captured and stored by Windows Phone devices. Its features include decoding and encoding JPEG images, applying filters and effects, cropping, rotating, and resizing. The Nokia Imaging SDK provides more than 50 premade filters and effects. You can not only apply effects such as sepia, cartoon, sketch, and so on, but also apply auto-enhance, brightness control, hue, saturation, and many more. The SDK has been specifically developed for mobile imaging, with speed and memory performance as key drivers.

This article discusses a prerelease version of the Nokia Imaging SDK. All related information is subject to change.

This article discusses:

- Basics of the Nokia Imaging SDK
- Project organization and preparation
- Building the UI
- Applying a real-time sketch effect
- Capturing and saving an image
- Integrating with the Windows Phone 8 media lens

Technologies discussed:

Windows Phone 8, Nokia Imaging SDK

Code download available at:

github.com/Srikar-Doddi/PaperPhoto

In this article, I'll demonstrate the use of the sketch effect through an example app called Paper Photo. The sketch effect is applied to the viewfinder stream in real time and provides users the ability to capture images and save them to the phone's camera roll. I developed this app using Visual Studio 2012, the Nokia Imaging SDK and the Windows Phone 8 SDK. I used a Nokia Lumia 920 smartphone to test it.

Nokia Imaging SDK Basics

The Nokia Imaging SDK was created by Nokia to give you full access to a powerful library of image-manipulation tools that help you create great imaging apps quickly and easily. As mentioned, this SDK was designed with high performance and low-memory usage in mind, which is important because Microsoft and Nokia are working to create next-generation devices such as the Nokia Lumia 1020. The SDK is already used in Nokia's own imaging application, Creative Studio. The SDK and the library are available free of charge (see the license agreement at bit.ly/130tVHJ), and the SDK currently supports only Windows Phone 8 apps.

The Nokia Imaging SDK offers the following features:

- A simple-to-use API that's available both from managed code as well as native code. This means the SDK is provided as a Windows Phone Runtime library and you can call the methods of the library from C#, Visual Basic or C++. The API comes with a range of classes and methods for various imaging tasks.
- RAJPEG technology to access image data without decoding a whole JPEG image first, which allows for blazingly fast previews, application of effects, cropping and rotation of high-resolution images.

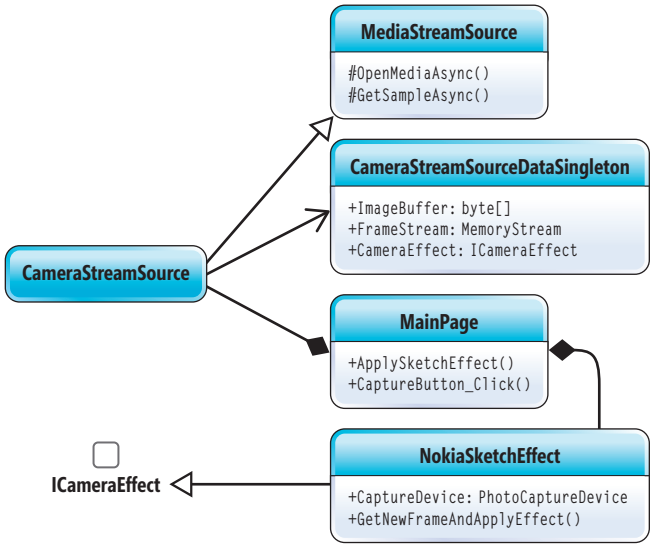


Figure 1 **The Class Diagram for the Paper Photo Project**

- More than 50 filters, effects and enhancements. Some of these enhancements allow you to programmatically adjust RGB levels, hue, saturation and brightness. You can also create custom effects and filters if needed. In addition to these filters and effects, the SDK also allows for cropping, rotating and resizing with unlimited undo functionality.
- Several fully featured sample apps allow you to explore the source code and discover the features of the SDK. The examples cover a range of features such as real-time manipulation of filter properties and application of several filter layers to photos.
- Rich documentation with quick-start guides, example projects, API reference guides and several documents that provide an overview of some of the core concepts needed to create imaging apps.

Getting Started

At a high level, I used the following APIs from the Nokia Imaging SDK for my sample app:

- Nokia.Graphics
- Nokia.Graphics.Imaging
- Nokia.InteropServices.WindowsRuntime

The Nokia.Graphics.Imaging API contains the core functionality of the Nokia Imaging SDK, which includes all the image filters, image effects, JPEG encoder and decoder. The Nokia.InteropServices.WindowsRuntime library is used internally and is a required library that needs to be referenced in your project. It contains a class called BufferFactory that's used for creating an instance of IBuffer. This SDK can be installed using the NuGet package manager in Visual Studio. You can find the package by searching for NokiaImagingSDK.

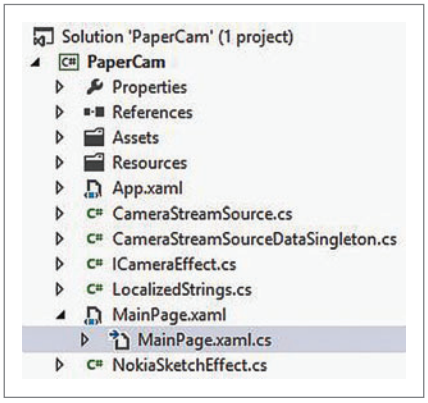


Figure 2 **Visual Studio Solution Structure**

The object graph of this app basically consists of three key classes, shown in **Figure 1**. The MainPage is your typical phone app page implemented by a XAML file and a C# counterpart. This MainPage class implements the app UI, which includes the MediaElement that displays the camera viewfinder with the sketch effect. The MainPage class also owns the instances of the two other key classes: CameraStreamSource and NokiaSketchEffect. The CameraStreamSource, derived from MediaStreamSource, provides the camera data, and the NokiaSketchEffect implements the sketch effect. The CameraStreamSource is a class implemented by Nokia and is provided to developers through the company's app samples outside of the Nokia Imaging SDK.

The object graph translates to the code organization shown in **Figure 2**.

Defining Your UI

The UI of this app is straightforward, as shown in **Figure 3**. The main page displays the image from a viewfinder and comes with an application bar that has a single button to capture the photo with the sketch mode effect.

Figure 4 shows the XAML markup for the main page using the Grid element to define the container used for showing the viewfinder with the sketch effect applied.

> The Nokia Imaging SDK was created to give you full access to a powerful library of image-manipulation tools that help you create great imaging apps.

The application bar is built using the markup shown in **Figure 5**. As you can see, it defines a capture button and an About menu item. The capture button is associated with an event handler to handle the click event.

Windows Phone uses a capabilities-driven security model. This model allows users to opt in to allow certain functionality to be activated. In my demo app, the following capabilities need to be enabled for users:

- ID_CAP_ISV_CAMERA: This provides access to the rear camera (primary) or the front-facing camera.
- ID_CAP_MEDIALIB_PHOTO: This provides read-only access to photos in the media library. It also gives an app the ability to save photos in the camera roll. I'll demonstrate this capability later in the article to save a photo to the camera roll after applying the sketch effect.

In terms of the hardware requirements, the app requires a rear-facing camera to function properly. The ID_REQ_REARCAMERA option is selected to prevent the app from installing on a phone without a rear camera.

Applying a Real-Time Sketch Effect to the Camera Viewfinder

The NokiaSketchEffect class is responsible for applying the sketch effect in real time to the camera viewfinder. Note the use of SketchMode.Gray to get the desired effect in my app. The GetSampleAsync function in the CameraStreamSource class is responsible for processing the camera buffer using the sketch effect, and it provides the media element with the buffer. The GetSampleAsync method uses the NokiaSketchEffect.GetNewFrameAndApplyEffect method to get the modified camera buffer. The code in **Figure 6** shows how it's implemented.

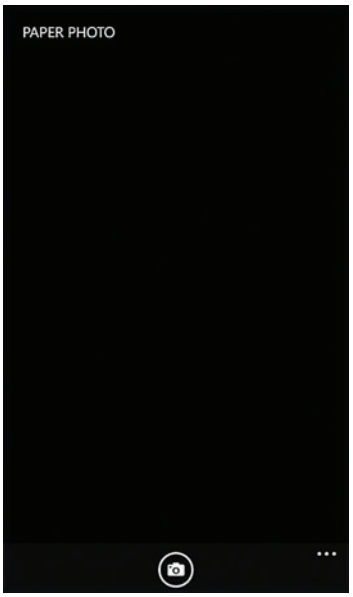


Figure 3 **The Paper Photo UI**

The captureDevice in the function is a private variable of the PhotoCaptureDevice class found in Windows.Phone.Media.Capture. The RenderToBitmapAsync function is provided by the Nokia Imaging SDK to support the asynchronous render of EditingSession to a bitmap. The EditingSession is an important class provided by the Nokia Imaging SDK to represent an image-processing editing session. The EditingSession object is the core of this SDK. In a typical workflow, the EditingSession is created from an image, filters or effects are added to this session, the EditingSession is then rendered to a bitmap or into a memory buffer, and finally the EditingSession is closed. The code in **Figure 6** shows how a new bitmap associated with the processedBuffer argument is created, the sketch effect is applied to it and then it's rendered into the new buffer. Finally, the CreateSketchFilter method produces the look of the sketched image. This function takes SketchMode—an enumeration—as an argument to pass the modes for the sketch filter. The two modes available are gray for sketch in grayscale and color for sketch in color. As you can see, I used SketchMode.Gray in the **Figure 6** code. This allows the Paper Photo app to produce images in grayscale.

Figure 4 **The XAML Markup for the Main Page**

```
<Grid x:Name="LayoutRoot" Background="Transparent">
  <Grid.RowDefinitions>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="*"/>
  </Grid.RowDefinitions>
  <Grid Grid.Row="1" Margin="8">
    <Grid x:Name="MediaElementContainer" Margin="0"></Grid>
      <StackPanel
        x:Name="TitlePanel"
        Grid.Row="0"
        Margin="12,17,0,28">
        <TextBlock
          Text="{Binding Path=LocalizedResources.ApplicationTitle,
          Source={StaticResource LocalizedStrings}}"
          Style="{StaticResource PhoneTextNormalStyle}"
          Margin="12,0"/>
      </StackPanel>
    </Grid>
  </Grid>
```

Capturing and Saving the Picture as a JPEG Image to the Camera Roll

So far I've shown how to apply the sketch effect in real time to the viewfinder. Now I'll look at capturing a picture from the viewfinder and then saving it to the camera roll on the device. The capture function first initiates autofocus and then captures a photo. Besides handling the capture functionality from the application bar, you can also handle the capture initiated by the hardware device trigger.

The code shown in **Figure 7** enables capture through the camera button. More specifically, the function enables or disables the hardware shutter release function. The CameraButtons class provides the events that are triggered by the device shutter buttons. One such event is the ShutterKeyHalfPressed event, triggered when the hardware shutter button is pressed and held for approximately 800 milliseconds. Another event is the ShutterKeyPressed event. This event occurs when the hardware shutter button receives a full press. The SetCameraButtonsEnabled function also handles the removal of these event handlers to help release the memory related to the camera.

Now I'll explain the capture process in detail. In the code shown in **Figure 8**, the camera object represents the photo capture device, the await camera.FocusAsync function initiates the autofocus and the StartCaptureAsync method captures a frame. But before you can capture the frame, you need to prepare the capture sequence. You do this by calling PrepareCaptureSequenceAsync. You're also creating

Figure 5 **The Application Bar Markup**

```
<phone:PhoneApplicationPage.ApplicationBar>
  <shell:ApplicationBar Opacity="0.4">
    <shell:ApplicationBarIconButton x:Name="CaptureButton"
      Text="Capture" IconUri="Assets/capture-button-icon.png"
      Click="CaptureButton_Click" IsEnabled="True" />
    <shell:ApplicationBar.MenuItems>
      <shell:ApplicationBarMenuItem
        Click="OnAboutPageButtonClicked" Text="about" />
    </shell:ApplicationBar.MenuItems>
  </shell:ApplicationBar>
</phone:PhoneApplicationPage.ApplicationBar>
```

Figure 6 **Getting the Modified Camera Buffer**

```
public async Task GetNewFrameAndApplyEffect(IBuffer processedBuffer)
{
  if (captureDevice == null)
  {
    return;
  }

  captureDevice.GetPreviewBufferArgb(cameraBitmap.Pixels);

  Bitmap outputBtm = new Bitmap(
    outputBufferSize,
    ColorMode.Bgra8888,
    (uint)outputBufferSize.Width * 4,
    processedBuffer);

  EditingSession session = new EditingSession(cameraBitmap.AsBitmap());
  session.AddFilter(FilterFactory.CreateSketchFilter(SketchMode.Gray));
  await session.RenderToBitmapAsync(outputBtm);
}
```

NEW HOSTING

DYNAMIC PRODUCTS, FLEXIBILITY, AND SUPPORT

FOR YOUR WEB PROJECTS

APPS
TOOLS
TECHNOLOGY
PACKAGES

1&1®

1and1.com

NEW HO

THE NEW PROFESSIO

We are Internet and eBusiness experts, with more than 6,000 specialists in 10 countries. The Internet has been our business for over 25 years, our enthusiasm and know-how has resulted in state-of-the-art technology, 19 million hosted websites in 5 high-performance data centers, and over 12 million customer accounts. See why so many are choosing 1&1 as their web hosting partner.

DOMAINS | E-MAIL | WEB HOSTING | eCOMMERCE | SERVERS

STING

NAL STANDARD

We know what professionals want – flexible solutions that are perfectly matched to their needs.

APPS

TOOLS

TECHNOLOGY

PACKAGES

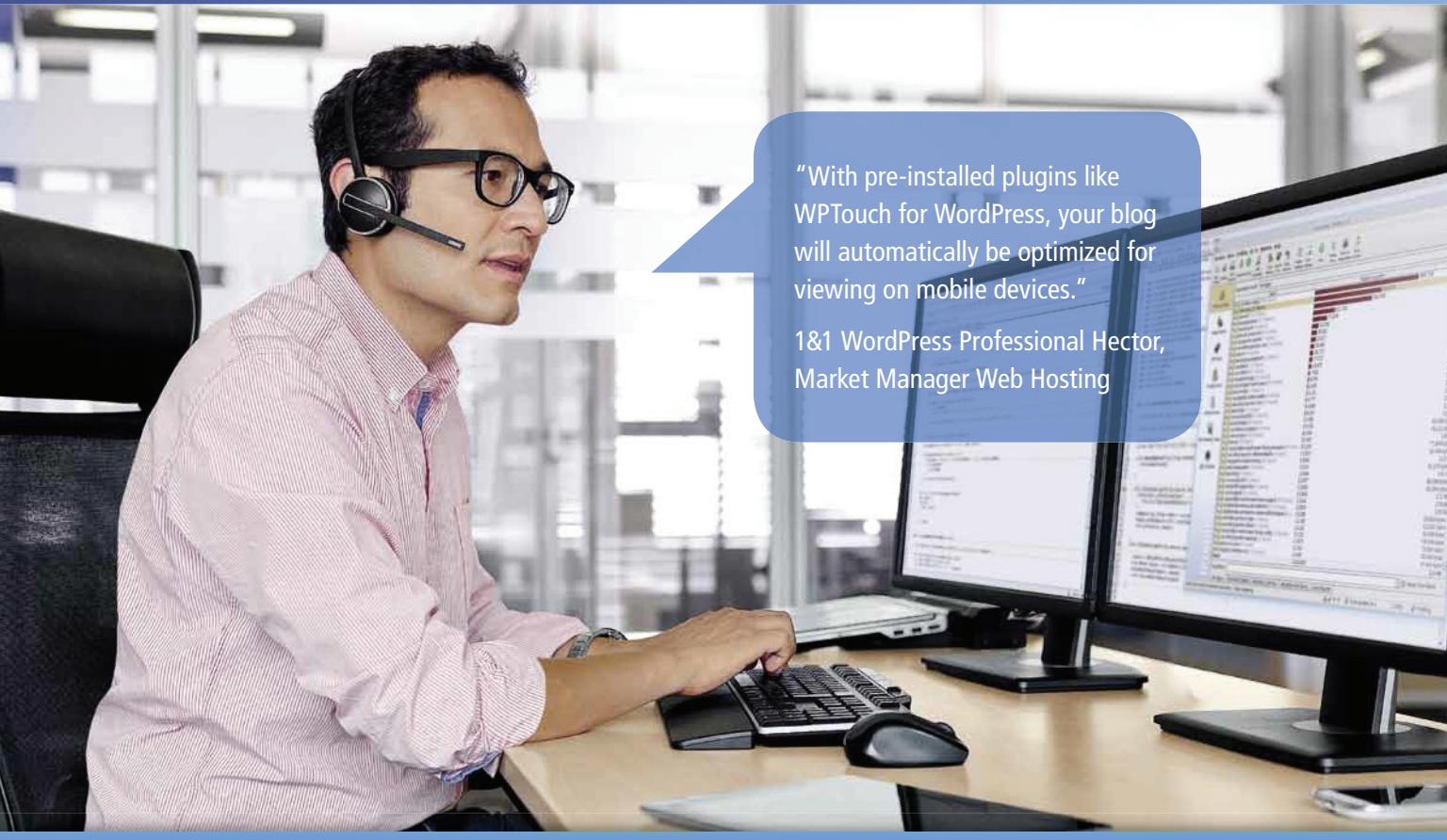
1&1®

Call **1 (877) 461-2631** or buy online

1and1.com

THE NEW 1&1

APP EXPERT CENTER

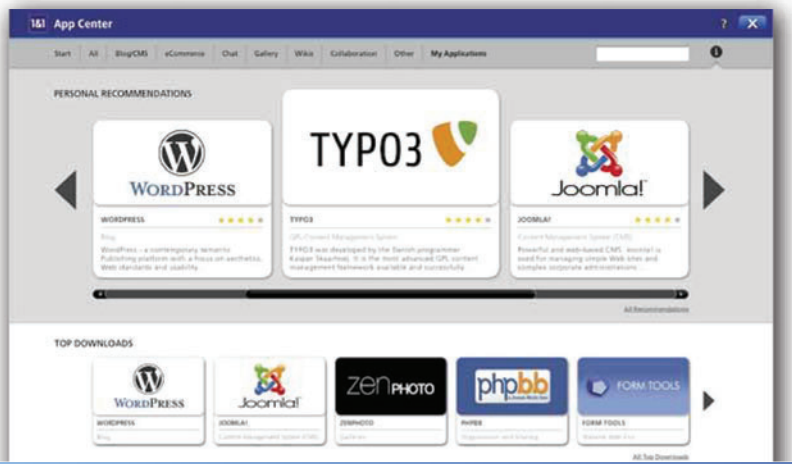


DOMAINS | E-MAIL | WEB HOSTING | eCOMMERCE | SERVERS

In our 1&1 App Expert Center, you will find the 40 most popular applications featuring useful plug-ins and one-click installation. Plus, get tips and tricks on how to get the most from each application.

EXPERTS ON TOP APPS

The 1&1 App Expert team provides support for the 5 most used apps and plug-ins, PHP and MySQL and support for every stage of your web project. Our experts will support you all the way!



YOU CHOOSE:

FREE MODE OR SAFE MODE?

If you want maximum freedom and control, then use Free Mode. You'll benefit from time-saving 1-click installation, as well as the ability to code plug-ins yourself at any time! We'll keep you informed regarding necessary updates and security patches so you can implement them yourself – in Free Mode, you're the boss!

If convenience and security are your top priorities, then use Safe Mode. While you'll only have access to the default plug-ins, we manage the apps for you, take care of all updates and security patches, and more. Plus, if you feel confident you always have the option of switching to Free Mode and personally taking control!

ALSO INCLUDED:

- **1&1 WEB APP PORTAL** with HTML code for over 100 popular widgets
- **GUARANTEED RESOURCES** for demanding applications

1&1®

Call **1 (877) 461-2631** or buy online

1and1.com

PROFESSIONAL

TOOLS

FOR YOUR WEB PROJECTS

PHP INCLUDED

Do you regularly use PHP? With 1&1 you'll have access to PHP 5.4 from Zend, the component-based framework for PHP, with additional support from the 1&1 App Center Experts.

1&1 MOBILE WEBSITE BUILDER

1&1 Mobile Website Builder automatically converts your website for mobile optimization to be viewed on smartphones and tablets. You can also manually edit all of your pages.

DOMAINS | E-MAIL | WEB HOSTING | eCOMMERCE | SERVERS

SOFTWARE INCLUDED!

ADDITIONALLY:

- **SOFTWARE: git-versioning** available via SSH
- **LINUX:** Web space recovery with up to 6 restore points at anytime, support for additional languages including Perl, Python and Ruby
- **WINDOWS:** ASP.NET 4.5, ASP MVC 3 and 4, PHP and Perl support for .NET Framework, dedicated app pools, and up to 10 .NET Applications.

1&1®

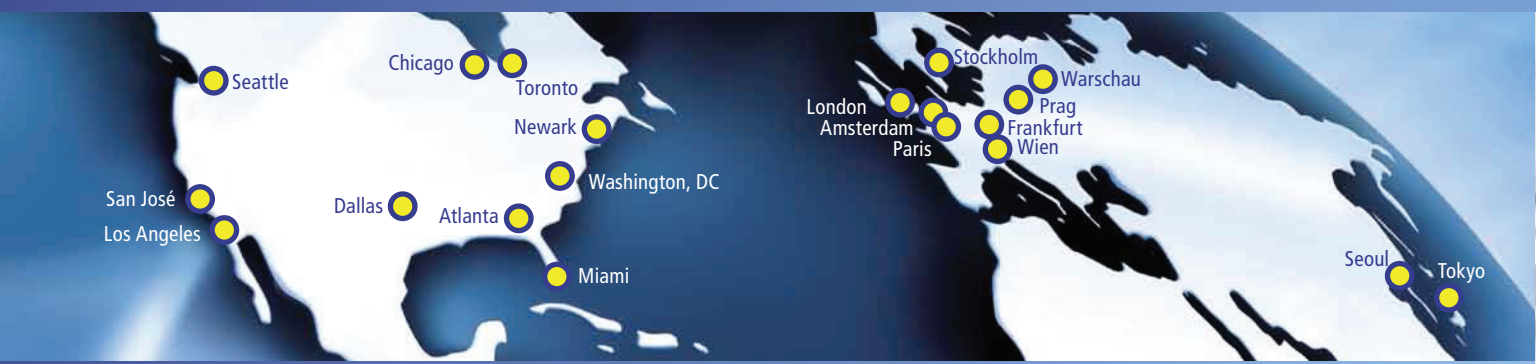
Call **1 (877) 461-2631** or buy online

1and1.com

STATE-OF-THE-ART

TECHNOL

The high-performance 1&1 Data Centers are among the safest and most efficient in the world. Redundant data centers provide the highest level of availability.



NEW CONTENT DELIVERY NETWORK

1&1 CDN (Content Delivery Network) provides maximum performance for your website. PoPs are distributed over 23 locations worldwide on different backbones. Through this network, the data of static website pages is stored closer to your visitors. This enables your visitors to get fast access to your website.

>300 GBIT/S CONNECTION

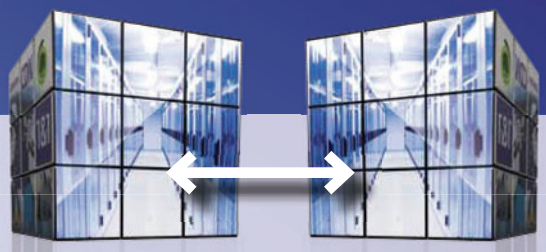
Our own external network connectivity backbone! To avoid data transfer congestion, 1&1 Data Centers are geo-redundant so that most important Internet nodes can be tethered, meaning that the fastest possible connection is automatically selected. You will also benefit from maximum uptime.

2 GB RAM GUARANTEED

Our top shared hosting package gives you the experience of a dedicated server – with 2 GB RAM for guaranteed performance! This makes demanding applications and large dynamic sites run smoothly with dedicated resources. Your site will also perform better during high volume visitor traffic.

DOMAINS | E-MAIL | WEB HOSTING | eCOMMERCE | SERVERS

OGY



"1&1 offers the highest standard of data security. With geo-redundancy, your data is stored simultaneously in two of our high-performance data centers in two separate locations. Our robust infrastructure ensures that your website offers the best possible performance."

1&1 Technology Specialist Stefan,
Head of Development Web Hosting & Servers

ALSO INCLUDES:

- **DAILY BACKUPS**
  For the complete infrastructure
- **PROACTIVE MONITORING**
  From 1&1 Experts
- **NEW SOFTWARE VERSIONS**
  With the latest features and security patches

1&1®

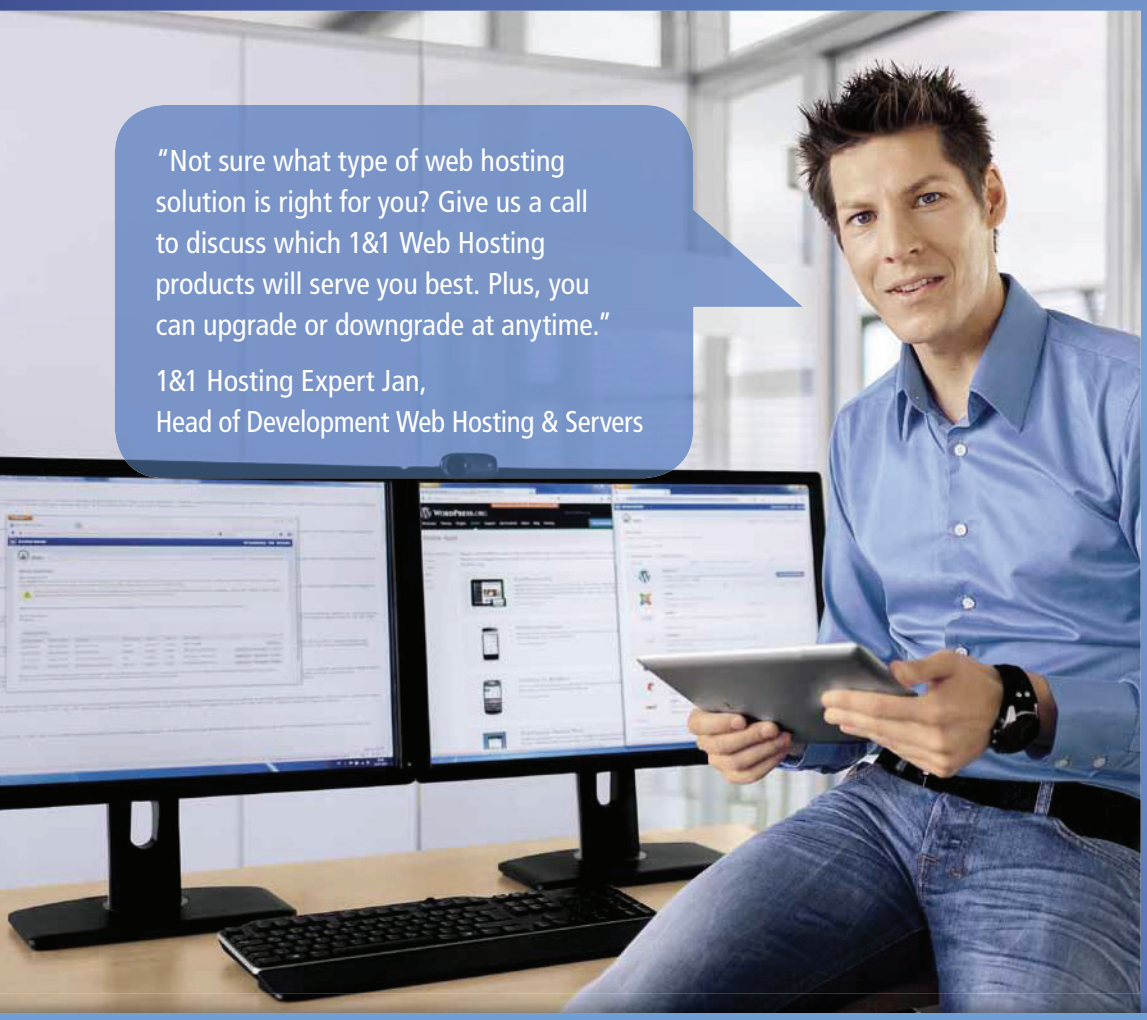
Call **1 (877) 461-2631** or buy online

1and1.com

1&1 NEW HOSTING

PACKAGES

TO FIT YOUR NEEDS



With 1&1, you'll always find the perfect package for your needs. Our website is constantly updated with current offers for virtual, cloud and dedicated servers, eCommerce and other eBusiness solutions.

DOMAINS | E-MAIL | WEB HOSTING | eCOMMERCE | SERVERS

* 1&1 Web Hosting packages come with a 30 day money back guarantee, no minimum contract term, and no setup fee. Prices in table reflect 36 month pre-payment option.

	Basic	Unlimited	Advanced
	One or more websites, high memory demands	Websites with interactive and dynamic content	Guaranteed performance, resource-intensive projects
Domains included (.com .net .org .biz .info)	1	1	1
No. of websites	1	UNLIMITED	UNLIMITED
Web space	100 GB	UNLIMITED	UNLIMITED
Traffic	UNLIMITED	UNLIMITED	UNLIMITED
E-mail accounts	100	UNLIMITED	UNLIMITED
Databases	20	UNLIMITED	UNLIMITED
APPS			
Click & Build Apps	✓	✓	✓
Web Apps	✓	✓	✓
Expert Support for top apps, PHP and MySQL	–	–	✓
Guaranteed Resources	–	–	**2 GB RAM**
TOOLS			
1&1 Mobile Website Builder	–	✓	✓
Premium software	NetObjects® Fusion® 2013 1&1 Edition	NetObjects® Fusion® 2013 1&1 Edition	NetObjects® Fusion® 2013, Adobe® Dreamweaver CS 5.5
PHP 5.4	✓	✓	✓
Git version control tool	✓	✓	✓
Perl, Python, Ruby	✓	✓	✓
INFRASTRUCTURE			
Maximum availability (Geo-redundancy)	✓	✓	✓
Redundant Network connectivity	> 300 Gbit/s	> 300 Gbit/s	> 300 Gbit/s
1&1 CDN powered by CloudFlare	–	✓	✓
1&1 Webspace Recovery	✓	✓	✓
Support	24/7	24/7	24/7 plus App Expert Support
	$1.99 ~~$5.99~~ per month*	**$3.99** ~~$8.99~~ per month*	**$5.99** ~~$14.99~~ per month*

1&1®

Call **1 (877) 461-2631** or buy online

1and1.com

THE 1&1 PRINCIPLES

THE PROFESSIONAL DIFFERENCE

1

TRIAL

. . . with 1&1 you can test your package knowing you have a 30 day money back guarantee. If you are not satisfied for any reason you can easily cancel to receive a refund.

MONTH

. . . short term payment options on request. If you don't want to commit for more than one month, then choose maximum flexibility with monthly billing, or save more with a longer pre-paid plan.

CLICK

. . . with just the click of a button you can upgrade or downgrade your hosting package. Change your package within the same product group once a month at no extra cost and only pay for what you use.

CALL

. . . all it takes is one call to speak with an expert 24/7. Our experts are always here for you.

SECURE CHOICE

. . . 1&1 Data Centers are state-of-the-art. We use geo-redundancy so that in the unlikely event of a failure, fast switching between our data centers provide maximum reliability . . . another reason millions trust 1&1.

October 2013

US104020300019

1&1®

MEMBER OF united internet

Call **1 (877) 461-2631** or buy online

1and1.com

a capture sequence object with one frame, as you can see from camera.CreateCaptureSequence(1). The value 1 indicates the number of frames that will be captured immediately after you initiate the capture. Finally, you also specify camera properties and photo settings using the KnownCameraPhotoProperties method. The LockedAutoFocusParameters is set to None, indicating that focus, exposure and white balance settings will adjust automatically prior to capture.

Finally, the following code shows how to save the captured photo to the camera roll:

```
MediaLibrary library = new MediaLibrary();

EditingSession session =
  new EditingSession(imageStream.GetWindowsRuntimeBuffer());

using (session)
{
  session.AddFilter(FilterFactory.CreateSketchFilter(SketchMode.Gray));
  IBuffer data = await session.RenderToJpegAsync();
  library.SavePictureToCameraRoll(FileNamePrefix
    + DateTime.Now.ToString() + ".jpg",
    data.AsStream());
}
```

The MediaLibrary class is provided by the Microsoft.XNA.Framework.Media API and is used to save photos to the phone's media library. The Microsoft.XNA.Framework.Media namespace contains classes to enumerate, play, and view songs, albums, playlists, and pictures. You then use the EditingSession class from the Nokia Imaging SDK to create a session from the compressed image buffer and then apply the sketch effect filter before saving the picture to the camera roll using the SavePictureToCameraRoll function.

Figure 7 **Enabling Capture Through the Camera Button**

```
private void SetCameraButtonsEnabled(bool enabled)
{
  if (enabled)
  {
    Microsoft.Devices.CameraButtons.ShutterKeyHalfPressed
      += ShutterKeyHalfPressed;
    Microsoft.Devices.CameraButtons.ShutterKeyPressed
      += ShutterKeyPressed;
  }
  else
  {
    Microsoft.Devices.CameraButtons.ShutterKeyHalfPressed
      -= ShutterKeyHalfPressed;
    Microsoft.Devices.CameraButtons.ShutterKeyPressed
      -= ShutterKeyPressed;
  }
}
```

Figure 8 **The Capture Process**

```
private async System.Threading.Tasks.Task Capture()
{
  try
  {
    await camera.FocusAsync();

    MemoryStream imageStream = new MemoryStream();
    imageStream.Seek(0, SeekOrigin.Begin);

    CameraCaptureSequence sequence = camera.CreateCaptureSequence(1);
    sequence.Frames[0].CaptureStream = imageStream.AsOutputStream();

    await camera.PrepareCaptureSequenceAsync(sequence);
    await sequence.StartCaptureAsync();

    camera.SetProperty(
      KnownCameraPhotoProperties.LockedAutoFocusParameters,
      AutoFocusParameters.None);
    ...
}
```

Figure 9 **A Regular Image Taken with the Built-in Camera App**

Figure 10 **An Image Taken Using the Paper Photo App Using the Sketch Effect**

Lens Picker Integration

In Windows Phone 8, you can create a rich media lens that opens from the built-in camera app and launches your app right into the viewfinder experience. In order to integrate with the lens experience, the app needs to register for the Camera_Capture_App extension. This extension declares to the OS that the Paper Photo app can display a viewfinder when it's launched from the lens picker. Extensions are specified in the WMAppManifest.xml file. You need to open this file with the text editor: Right-click on the file and choose "Open With… | Text Editor." Just after the Tokens element, inside the Extensions element, the lens extension is specified with the following Extension element:

```
<Extension ExtensionName="Camera_Capture_App"
  ConsumerID="{5B04B775-356B-4AA0-AAF8-6491FFEA5631}"
  TaskID="_default" />
```

Now your app is complete with lens integration and can save pictures right into the camera roll on the device. The image in **Figure 9** was taken with the built-in camera app, and the image in **Figure 10** was taken with the Paper Photo app using the sketch effect.

Add Your Own Features

In this article, I explored photo capture APIs from Microsoft and the brand-new Nokia Imaging SDK to create a rich imaging experience. I applied the sketch effect to the viewfinder in real time and then captured the photo and saved it on the device camera roll. As you can see, there are tons of features that you can now use thanks to the Nokia Imaging SDK, which makes it easy to build apps such as this one. A few minor details were omitted for the sake of brevity, but you can refer to the code download (github.com/Srikar-Doddi/PaperPhoto) for a complete understanding. This is just one example of using the Nokia Imaging SDK, but there are lots of other features you can add and the possibilities are unlimited. I hope you use this code as a starting point and add more features to this app. ■

Srikar Doddi *is the executive director of engineering at Kaplan Test Prep. You can find his writings on Medium at medium.com/@SrikarDoddi. He is also the creator of the Simplist, DateTileScheduler and Paper Photo apps for Windows Phone 8 and the Prompter and Dabble apps for Windows 8. You can reach Doddi via e-mail at srikar.doddi@gmail.com.*

Thanks to the following technical expert for reviewing this article: *Lance McCarthy (Nokia)*

LIVE EVENT

AMPLIFY YOUR KNOWLEDGE

Live! 360 is back to turn your conference experience up to 11. Spanning 5 days at the Royal Pacific Resort in sunny Orlando, Live! 360 gives you the ultimate IT and Developer line-up, offering hundreds of sessions on the most relevant topics affecting your business today.

SUPPORTED BY Microsoft SharePoint Microsoft SQL Server 2012 Visual Studio msdn magazine Visual Studio MAGAZINE

PRODUCED BY 1105 MEDIA

- 5 Days.
- 4 Events.
- 20 Tracks.
- 175+ Sessions.
- The Ultimate IT and Developer Line-up.

REGISTER BY OCTOBER 9 & SAVE $300!

Use Promo Code LIVEOCT4

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

ORLANDO | **NOVEMBER 18-22, 2013**

Royal Pacific Resort at Universal Orlando

LIVE! 360

IT EVENTS WITH PERSPECTIVE

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

Calling all .NET Developers! We've got your ticket to Code for the final stop on the Visual Studio Live! 2013 Tour. **vslive.com/orlando**

SharePoint LIVE!

TRAINING FOR COLLABORATION

Collaborate and Listen. SharePoint Live! returns to rock its newest release, SharePoint 2013. **splive360.com**

SQL Server LIVE!

TRAINING FOR DBAs AND IT PROS

Bringing SQL Server to Your World. Fine tune your skills to solve your biggest data challenges at SQL Server Live!. **sqllive360.com**

Modern Apps LIVE!

MODERN APPS FROM START TO FINISH

The Future of Software Development is Back, Modern Apps Live! will break down the latest techniques in low cost, high value application development. **modernappslive.com**

TURN PAGE FOR MORE EVENT DETAILS ▶

LIVE! 360
IT EVENTS WITH PERSPECTIVE

5 DAYS. 4 EVENTS.

Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS

Intense Take-Home Training for Developers, Software Architects and Designers

TRACK TOPICS:
- Azure / Cloud Computing
- Mobile Development
- Data Management
- Visual Studio / .NET Development
- Web and JavaScript Development
- Windows 8 / WinRT

SQL Server LIVE!
TRAINING FOR DBAs AND IT PROS

Building, Developing and Managing SQL Server

TRACK TOPICS:
- SQL Server Administration and Maintenance
- SQL Server Performance Tuning and Optimization
- SQL Server Tools, Tricks, and Techniques
- SQL Server Features and Components
- BI, Big Data, and Data Visualization
- SQL Server for the Developer
- SQL Server in the Cloud

CONNECT WITH LIVE!360

twitter.com/@live360events facebook.com – Search "Live 360" linkedin.com – Join the "Live 360" group!

20 TRACKS. 1 LOW PRICE!

SharePoint LIVE!

TRAINING FOR COLLABORATION

No-Hype, Practical, Independent SharePoint Training

TRACK TOPICS:

- Keys to SharePoint Success: Strategy, Governance, Adoption
- SharePoint Infrastructure Management and Administration
- High-Value SharePoint Workloads: Social, Search, BI, and Business Process Automation
- Information and Content Management: Documents, Records, and Web
- Developing Apps and Solutions for SharePoint
- SharePoint, Office 365 and the Cloud

Modern Apps LIVE!

MODERN APPS FROM START TO FINISH

The Future of Software Development is Back!

LEARN HOW TO BUILD MODERN APPLICATIONS:

- Better
- Quickly
- Cheaply

What sets Modern Apps Live! apart is the singular topic focus; sessions build on each other as the conference progresses, leaving you with a holistic understanding of modern applications.

LIVE! 360

IT EVENTS WITH PERSPECTIVE

ORLANDO • NOVEMBER • 18-22, 2013

- Forward-thinking, educational events
- Featuring what's now, new and next for IT Pros AND Developers
- Includes knowledge transfer, networking and training

REGISTER BY OCTOBER 9 & SAVE $300!

Use Promo Code LIVEOCT4

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

live360events.com

Adaptive Access Layers + Dependency Injection = Productivity

Ulrik Born

One of the toughest challenges when managing and developing the source code of a complex enterprise solution is to ensure the codebase remains consistent, intuitive, and highly testable while being maintained and extended by multiple teams within a large software development department. The core problem is that developers typically have a number of rules and guidelines to follow and a set of utility libraries to use, which tend to cause their solutions to become bigger and more complex. This is because they end up tweaking the ideal, intuitive business logic implementation to make it adhere to rules and fit fixed APIs. This generally means more work for the developers, more bugs, less standardization, less reuse, and reduced overall productivity and quality.

I'm a senior developer for a leading online investment bank and have observed how these challenges can limit productivity. This article is a case study that presents how our development team has analyzed and overcome the challenges by innovative use of runtime code generation and Dependency Injection (DI). You might not agree with some of our team's design choices, but I believe you'll agree they represent a fresh and efficient way of addressing some common architectural challenges.

This article discusses:

- Why Adaptive Access Layers (AAL) were developed
- Integration of AAL with Dependency Injection
- Implementing AAL
- Reporting and checking
- Benefits of using AAL

Technologies discussed:

Adaptive Access Layers, Windows Communication Foundation, SQL Server, Dependency Injection, Windows Services

My company has a big in-house software development department (working on two continents) that constantly maintains and extends our huge Microsoft .NET Framework codebase. Our codebase is focused on numerous mission-critical Windows services that make up the high-performance, low-latency trading system hosted in our datacenters. We have a number of platform teams that guard the codebase and runtime environment, plus many project teams that continuously (and in parallel) improve and extend the system.

I've worked on platform teams for several years and experienced the downside of an inconsistent and overly complex codebase during numerous reviews and support cases. Two years ago, we decided to address these issues, and I found the following problems:

- We had way too many solutions to the same fundamental problems. A good example is that most of our Windows services had their own unique way of combining the various APIs into a simple service with proper support for logging, tracing, database access and so on.
- Our business logic implementations were either simple—but not unit-testable and too naïve, not adhering to guidelines—or overly complex due to a lot of plumbing code.

A common example: simple code that worked directly on the .NET SQL Server API versus complex code that spent far more lines on trivial plumbing to support automatic retries, caching and so on than on the real business logic.

- We had utility libraries supporting most of our architectural principles and coding guidelines, but they were implemented in several different styles and evolved independently. So even when using them as dictated by the guidelines, each feature solution ended up having a huge footprint in terms of referenced assemblies and exposure toward API changes. This in turn made it a complex task in itself to put a new feature in production and also made it difficult to update the utility libraries.
- The overall set of guidelines and rules to apply and utilities to use was simply so big that only our most experienced developers had a fair chance of understanding everything, and the entry barrier for new developers was extremely high. This meant a lot of nonstandard code was written, either to be discarded later or to reach production with increased inconsistency.
- Several of our core Windows services had central "registration points" where all project teams had to touch the same code—for example, a big switch statement dispatching commands or jobs. This made it nontrivial to merge code into our main branch.

Natrually, these problems were neither new nor unique to us, and a number of well-established design patterns describe how to address such problems:

- The façade pattern hides all the details of accessing a complex resource behind a simple *access layer* interface. This facilitates clean and testable business logic implementations where external resources can be easily mocked out in tests.
- DI—or Inversion of Control (IoC) containers—allows components to be loosely coupled and therefore easier to extend, maintain and combine. This technique also makes it easy to mock out selected components and thereby increase testability.
- Well-designed utility library APIs don't force the consuming code to be tweaked; instead, they support intuitive implementation.

We'd been aware of these patterns for years and had also applied them all in various forms throughout our codebase. But a few fundamental problems had significantly limited the success of these patterns. First, the façade pattern doesn't eliminate the need for a lot of plumbing code—it just moves it into another class and generally just means more work for the developer. Second, unless the DI container automatically discovers its components at run time (for example, via attributes), it will still require a central registration and will in reality just introduce an additional layer into the implementation. And, finally, it's costly and extremely difficult to design and implement APIs that are intuitive, flexible and useful at the same time.

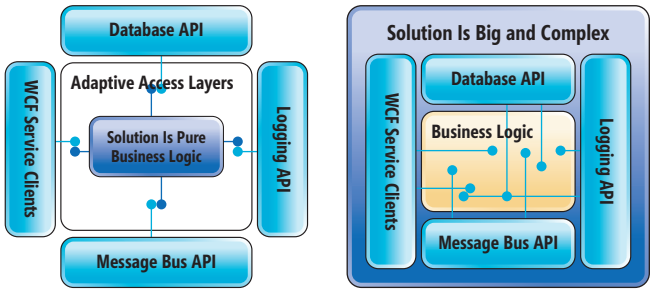


Figure 1 **Adaptive Access Layers (Depicted on the Left) Dramatically Reduce Solution Complexity and Footprint**

A key feature of the AAL technique is that it gives us a common central place to implement our best practices and guidelines without polluting the business logic code.

Why We Created Adaptive Access Layers

After a number of brainstorming sessions, we came up with a single flexible and powerful solution to all of these problems. The basic idea is to hide all APIs behind attributed access-layer interfaces and build an implementation engine that can implement such interfaces at run time in a way that follows all rules and guidelines. We call this technique Adaptive Access Layers (AAL) because each solution defines the access-layer interfaces it needs in a highly flexible way. We've combined the AAL implementation engine with the open source, attribute-driven Autofac DI container and achieved a flexible Windows service framework that makes clean, intuitive and testable implementations the easiest option. **Figure 1** illustrates how the size, footprint and complexity of a solution are reduced dramatically when AAL is used to decouple the core logic implementation from all the surrounding libraries and APIs. The blue boxes represent the footprints of a single solution, implemented with AAL (on the left) and without (on the right).

A key feature of the AAL technique is that it gives us a common central place to implement our best practices and guidelines without polluting the business logic code. In that respect, AAL is similar to aspect-oriented programming (AOP), various interception features and proxy techniques. The main difference is that AAL hides the underlying APIs from the consuming business logic code, whereas the other techniques still expose them and thus increase solution footprint significantly.

To illustrate the idea, I'll discuss a simple access layer between some business logic and the standard Windows event log. Consider an order registration service that registers incoming orders in a database. If the database call fails, the service must write an error to the event log.

In the classic approach, this could involve a call to the .NET EventLog.WriteEntry method and could look like the code in **Figure 2**. This approach isn't optimal for two reasons. First, it isn't well-suited for unit testing, as the test would have to inspect the event log on

the machine running the unit tests to validate that an entry with the correct text was actually written. And second, four lines of trivial plumbing code will "pollute" a core part of the business logic.

Both of these problems are addressed by introducing an AAL interface between the business logic and the underlying EventLog class. Such a layer is illustrated in the following code:

```
[EventLogContract("OrderService")]
public interface IOrderServiceEventLog
{
  [EventEntryContract(1000, EventLogEntryType.Error,
    "Order {0} not reg due to error: {1}"]
  void OrderRegistrationFailed(int orderId, string message);
}
```

The layer is defined by the attributed interface IOrderServiceEventLog that's implemented via a dynamic class by the implementation engine at run time. The interface itself has the [EventLogContract] attribute to allow the implementation engine to recognize it as an event log access layer. The single parameter is the name of the event log to which to target. There are no restrictions on the name of the interface or the number of methods on it. Each method must return void (there's no meaningful return value when writing information to the event log) and have the [EventEntryContract] attribute. The attribute takes all the fixed metadata input (id, severity and formatting) as parameters such that this no longer needs to be in the business logic.

Using the access-layer interface, the business logic from **Figure 2** becomes a lot smaller and clearer:

```
public OrderConfirmation RegisterOrder(Order order)
{
  try
  {
    // Call database to register order and return confirmation
  }
  catch (Exception ex)
  {
    _logLayer.OrderRegistrationFailed(order.Id, ex.Message);
  }
}
```

The sample RegisterOrder method is now simple, highly readable and a lot more testable, because validation no longer requires inspection of the event log but instead just a small mock class implementing the access-layer interface. Another advantage is that the IOrderServiceEventLog interface can map the full event log interaction by the entire order service and thus provide a simple yet complete overview of what event log entries the system writes.

(Note: As a brief aside, the recent Semantic Logging Application Block [SLAB] over the new .NET 4.5 EventSource class embraces the same ideas of moving metadata from code into attributes and exposing customized, strongly typed logging methods instead of a few general-purpose methods. To use SLAB, developers must implement a custom class derived from the EventSource class and use this class throughout the codebase. I believe our approach is as powerful as SLAB but easier to use, as it only requires developers to define the interface and not the class implementation. A key feature of the EventSource class is that it supports structured event logging via a configurable set of sinks. Our access-layer implementation doesn't currently support structured logging but could easily be extended to do so because it has access to the structured information via the access-layer method parameters.)

Figure 2 **Classic Event Log Access**

```
public OrderConfirmation RegisterOrder(Order order)
{
  try
  {
    // Call database to register order and return confirmation
  }
  catch (Exception ex)
  {
    string msg = string.Format("Order {0} not registered due to error: {1}",
      order.OrderId,
      ex.Message);
    _eventLog.WriteEntry(msg, 1000, EventLogEntryType.Error);
  }
}
```

Database access is a central part of most enterprise systems, including ours.

I haven't yet considered the real body of the RegisterOrder method, namely the call to some SQL Server database stored procedure to persist the order for further processing. If my team implemented this using the .NET SqlClient API, it would be at least 10 lines of trivial code to create a SqlConnection instance and SqlCommand instance, populate the command with parameters from the Order properties, execute the command and read back the result set. If we were to meet additional requirements such as automatic retries in the case of database deadlocks or time-outs, we could easily end up with 15 to 20 lines of code just to make a fairly simple call. And all of this would be required just because the target of the call happened to be a stored procedure rather than an in-process .NET method. From a business logic perspective, there's absolutely no reason why our core implementation should be so cluttered and complex just because the processing crosses from one system to another.

By introducing an adaptive database access layer similar to the event log access layer, we can implement the body as simple and testable:

```
public OrderConfirmation RegisterOrder(Order order)
{
  try
  {
    return _ordersDbLayer.RegisterOrder(order);
  }
  catch (Exception ex)
  {
    _logLayer.OrderRegistrationFailed(order.Id, ex.Message);
  }
}
```

So far, I've illustrated the ideas, flexibility and power of AAL. I'll now move on with a more detailed walk-through of the access layers we've developed and found useful. I'll start with the aforementioned database access layer.

Database Access Layers Database access is a central part of most enterprise systems, including ours. Being a key facilitator of serious online trading of financial instruments, we have to meet some strict performance and security requirements demanded by our clients and the financial authorities, and are therefore forced to safeguard our databases carefully. We generally do this by only doing database access via stored procedures, as that lets us apply fine-grained security rules and review all database queries for performance and server load before they hit our production systems.

ALEXSYS TEAM®

WORKFLOW APPLICATIONS | HELP DESK | BUG TRACKING

MADE EASY!

Alexsys Team® offers ***flexible task management*** software for Windows, Web, and Mobile Devices. Track whatever you need to get the job done - anywhere, anytime on practically any device!

Thousands are using Alexsys Team® to create workflow solutions and web apps - without coding!
Fortune 500 companies, state, local, DOD, and other federal agencies use Team to manage their tasks. Easily tailor Team to meet your exact requirements - even if they change daily, weekly, or even every minute.

Alexsys Team® Features Include:

- Form and Database customization
- Custom workflows
- Role-based security
- Adaptive Rules Engine
- DOD CAC card support
- Time recording
- Automated Escalations, Notifications, and SOAP Messaging
- Supports MS-SQL, MySQL, and Orcale databases

Our renowned tech support team is here to make you and your team a success. Don't have enough resources? Our professional services staff has helped companies large and small use Alexsys Team® at a fraction of the cost of those big consulting firms.

Find out yourself:
Free Trial and Single User FreePack™ available at Alexcorp.com

FIND OUT MORE!

1-888-880-ALEX (2539)
ALEXCORP.COM

We've carefully evaluated whether object-relational mapping (ORM) tools such as the Entity Framework could help us achieve simpler and more testable code without moving away from stored procedures. Our conclusion was that the Entity Framework is an extremely appealing solution, but it relies heavily on being able to compose and execute complex SQL statements at run time. It can map stored procedures, but when limited to mapping only stored procedures, it loses most of its benefits. For that reason, we decided to implement our own database-access framework as an adaptive database access layer.

Our implementation supports stored procedure calls, selects on views and effective mass inserts via the SQL Server bulk copy functionality. It can map input data directly from Data Transfer Object (DTO) class properties to stored procedure parameters and can likewise map result set columns into class properties. This facilitates a clear and direct syntax when doing database access in .NET code.

The following code shows a simple layer that suits the sample order registration service:

```
[DatabaseContract("Orders")]
public interface IOrdersDatabase
{
  [StoredProcedureContract("dbo.RegisterOrder", Returns=ReturnOption.SingleRow)]
  OrderConfirmation RegisterOrder(Order order);
}
```

This code maps a single stored procedure and turns the single row in the result set into an OrderConfirmation instance initialized from the result set columns. The parameters of the mapped stored procedure are set from the properties of the given Order instance. This mapping behavior is defined in the [StoredProcedureContract] attribute and thus is no longer required in the business logic implementation, making that clear and readable.

Our implementation supports stored procedure calls, selects on views and effective mass inserts via the SQL Server bulk copy functionality.

We've implemented some quite advanced features in the database access layer because we concluded it's a simple and efficient way of offering standard functionality to our developers without restricting their freedom to implement their business logic in the most natural and intuitive way.

One of the supported features is seamless support for bulk inserting rows via the SQL bulk copy functionality. Our support allows our developers to define a simple method that takes an enumerable collection of a DTO class representing the rows to insert as input. The access layer handles all the details and thereby relieves the business logic of 15 to 20 lines of complex, database-centric code. This bulk copy support is a perfect example of a conceptually simple operation—to insert rows into a table in an efficient way—that normally ends up being rather complex to implement simply because the underlying .NET Framework SqlBulkCopy class happens to work on an IDataReader rather than directly on our DTO class.

The database access layer was the first one we implemented, and it's been a huge success from the beginning. Our experience is that we write fewer and simpler lines of code with it and our solutions naturally become highly unit testable. Based on these positive results, we quickly realized we could benefit from introducing AAL between our business logic code and several other external resources.

Service Access Layers Our trading system implementation is highly service-oriented, and robust inter-service communication is essential for us. Our standard protocol is Windows Communication Foundation (WCF) and we have a lot of code focused on making WCF calls.

Most of these implementations follow the same overall pattern. First, the addresses of the endpoints are resolved (we typically run our services either in active-active or active-passive setups). Then the ChannelFactory .NET class is used to create a channel class implementation on which the desired method is invoked. If the method succeeds, the channel is closed and disposed, but if it fails, the exception has to be inspected. In some cases it makes sense to retry the method on the same endpoint, while in other scenarios it's better to do an automatic failover and retry on one of the other available endpoints. On top of this, we often want to quarantine a failed endpoint for a short time so as to not overload it with connection attempts and failing method calls.

It's far from trivial to write a correct implementation of this pattern, and it can easily take 10 to 15 lines of code. And again, this complexity is introduced only because the business logic we need to call happens to be hosted in another service and not in-process. We've implemented an adaptive service access layer to eliminate this complexity and make it as simple and safe to call a remote method as it is to call an in-process method.

The principles and workflow are identical to that of the database access layer. The developer writes an attributed interface that maps only the methods she needs to call, and our implementation engine creates a runtime type that implements the interface with the best practice behavior as specified in the attributes.

The following code shows a small service access layer that maps a single method:

```
[ServiceAccessLayer(typeof(IStatisticsSvc), "net.tcp", "STATISTICS_SVC")]
public interface ISalesStatistics
{
  [ServiceOperationContract("GetTopSellingItems")]
  Product[] GetTopProducts(int productCategory);
}
```

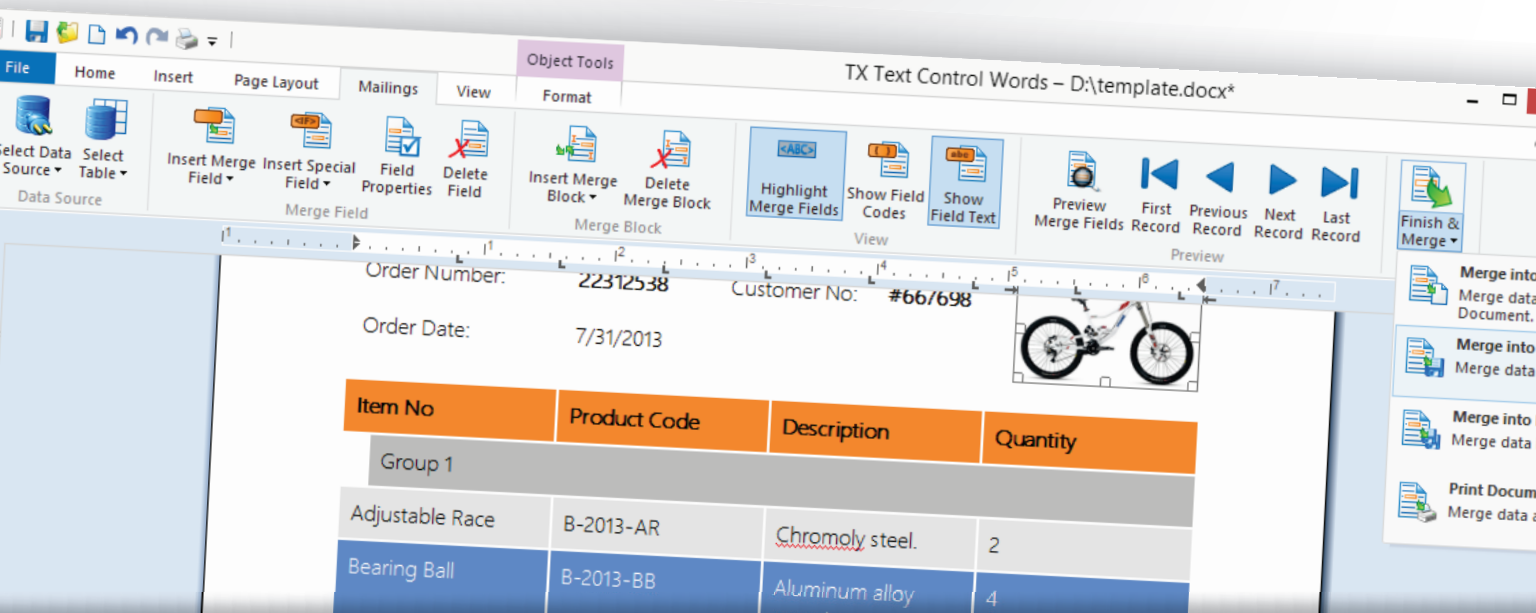
The interface attribute identifies the underlying plain [ServiceContract] attributed interface (to be used with the inner call to ChannelFactory), the protocol to use and the id of the service to call. The latter is used as a key into our service locator to resolve the actual endpoint addresses at call time. The access layer will by default use the default WCF binding for the given protocol, but this can be customized by setting additional properties on the [ServiceAccessLayer] attribute.

The single parameter to the ServiceOperationContract is the action verb that identifies the mapped method in the underlying WCF service contract. Other optional parameters to the attribute specify whether service call results shall be cached and whether

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

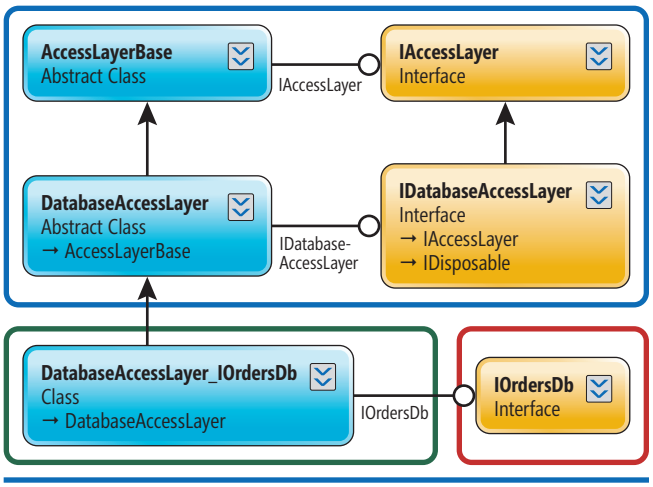


Figure 3 **An Access-Layer Implementation Outline**

it's always safe to automatically failover the operation even if the first WCF endpoint call fails after code has been executed on the target service.

Other Access Layers We've also built similar AAL for trace files, performance counters and our message bus. They're all based on the same principles illustrated by the preceding examples—namely, to allow the business logic to express its resource access in the simplest possible way by moving all metadata into attributes.

Dependency Injection Integration

With access layers, our developers no longer need to implement a lot of trivial plumbing code, but there still must be a way to invoke the AAL implementation engine to obtain an instance of the runtime implemented type whenever the mapped external resource has to be called. The implementation engine can be called directly, but that would go against our principles of keeping business logic clean and testable.

We've addressed this problem by registering our implementation engine as a *dynamic registration source* with Autofac such that it gets called whenever Autofac can't resolve a dependency with any of the static registrations. In this case, Autofac will ask the implementation engine whether it can resolve a given combination of type and id. The engine will inspect the type and provide an instance of the type if the type is an attributed access-layer interface.

With this in place, we've established an environment where business logic implementations simply can declare their access-layer interface types and take them as parameters (for example, in class constructors) and then trust that the DI container will be able to resolve these parameters by invoking the implementation engine behind the scenes. Such implementations will naturally work on interfaces and be easy to test as it only takes a few mock classes to implement these interfaces.

Implementation

All of our access layers are implemented using the same techniques. The overall idea is to implement all of the functionality in plain C# code in an abstract base class and then only use emit to generate a thin class that derives from the base class and implements the interface. The emitted body of each interface method simply forwards the execution to a general Execute method in the base class.

The signature of this general method is:

```
object Execute(Attribute, MethodInfo, object[], TAttributeData)
```

The first parameter is the method attribute of the access-layer interface method from which the Execute method is called. It generally holds all the metadata (for example, stored procedure name, retry specification and so on) needed for the Execute method to provide the correct runtime behavior.

The second parameter is the reflected MethodInfo instance for the interface method. It holds complete information about the implemented method—including the types and names of method parameters—and is used by the Execute method to interpret the third parameter. It holds the values of all parameters to the current interface method call. The Execute method typically forwards these values into the underlying resource API, for example, as stored procedure parameters.

The fourth parameter is a custom type that holds fixed data to be used at every invocation of the method in order to make it as efficient as possible. The fixed data are initialized once (by a method in the abstract base class) when the engine implements the runtime class. Our database access layers use this feature to inspect stored procedures once only and prepare a SqlCommand template ready for use when the method is invoked.

The Attribute and MethodInfo parameters passed to the Execute method are also reflected only once and reused in every method invocation, again to minimize the per-call overhead.

The return value of Execute is used as the return value for the implemented interface method.

This structure is quite simple and has turned out to be both flexible and powerful. We've reused it in all our access layers via an abstract, common base AccessLayerBase class. It implements all the required logic to inspect an attributed interface and drive the process of emitting a new runtime class. Each access-layer category has its own specialized abstract base class derived from AccessLayerBase. It holds the actual implementation of accessing the external resource, for example, making a stored procedure call according to all our best practices. **Figure 3** shows the implementation class hierarchy for a sample database access-layer interface. The blue section is the AAL framework; the red section is the attributed interface defined by the business logic feature solution; and the green section is the runtime class emitted by the AAL implementation engine.

Figure 3 also illustrates how we've let the base classes implement a set of public interfaces (deriving from IAccessLayer) to expose key behavioral information. This isn't intended to be used by business logic implementations but rather by infrastructure logic—for example, to track whenever a stored procedure invocation fails.

These access-layer interfaces are also useful in the few special cases where business or technical requirements demand that access to the underlying resource behind the access layer is done in a way not fully supported by AAL. With these interfaces, our developers can use AAL but intercept and adjust the underlying operations to meet special requirements. A good example of this is the IDatabaseAccessLayer.ExecutingCommand event. This is raised

Extreme Performance & Linear Scalability

Remove data storage and database performance bottlenecks and scale your applications to extreme transaction processing (XTP). NCache lets you cache data in memory and reduce expensive database trips. It also scales linearly by letting you add inexpensive cache servers at runtime.

Enterprise 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 & image merge/minify

Runtime Data Sharing

- Powerful event notifications for pub/sub data sharing

Download a 60-day FREE trial today!

FREE EXPRESS EDITION

NCache™

Distributed Cache for .NET & Java

www.alachisoft.com

1-800-253-8195

right before a SqlCommand is executed and allows us to customize it by altering things such as time-out values or parameters.

Reporting and Checking

The attributed AAL interfaces of a business logic solution also allow us to reflect the compiled binaries at build time and extract a number of useful reports. Our team has incorporated this in our Team Foundation Server (TFS) builds such that every build output now includes a few informative, small XML files.

Build-Time Reporting The database access layer reports the complete list of all stored procedures, views and bulk inserts being accessed. We use this to simplify reviews and to check that all required database objects have been properly deployed and configured before we release the business logic.

Likewise, our event log access layer reports the full list of event log entries a service can generate. Our post-build steps take this information and transform it into a management pack for our Microsoft System Center Operations Manager production environment surveillance. This is smart because it ensures that Operations Manager is always up-to-date with proper information on how to best handle production issues.

> Of course, the challenge of having a widely used and highly versatile platform is that it can become a single point of failure.

Automated Microsoft Installer Packages We've applied the same reflection techniques to harvest valuable input to the Microsoft Installer (MSI) packages we generate for our Windows services as the final step in our TFS builds. A key point for these packages is to install and configure event log and performance counters to ensure they match the business logic being deployed. The build extracts the event log names and performance counter definitions from the binaries and automatically generates an MSI package that installs these names and definitions.

Runtime Checking One of the most common errors reported from our production environment used to be that a service had attempted to call a stored procedure that didn't exist or existed with the wrong signature on the production database. These kinds of errors happened because we missed the deployment of all the required database objects when we deployed a Windows service to production. The critical issue here wasn't the missing deployment itself, as that could be fixed fairly easily, but more the fact that the error didn't happen during deployment but typically later on when the stored procedure was first called during business hours. We've used the reflection-based list of all accessed database objects to address this problem by letting our Windows services validate the existence and validity of all objects during service startup. The service simply runs through the list of objects, then queries the database for each one to check that it will be able to access the object when needed. This way, we've moved all such errors from business hours to deployment time, when it's a lot safer and easier to fix them.

I've listed these additional usages to illustrate a key benefit of AAL. With almost-complete information about service behavior easily available via reflection, a whole new dimension of intelligent reporting, building, automation and monitoring opens up. Our team has harvested a few of these benefits so far, but we see a number of additional interesting applications ahead.

Productivity and Quality

AAL, designed and implemented over the past two years, has proven to be an extremely powerful facilitator of higher developer productivity and increased solution quality for our company's dev team. We've reduced our costs for preparing a new Windows service from weeks to hours and for extending existing services from days to minutes. This has improved our agility and thereby made it cheaper for us to develop our customer offerings.

Our access layers are suitable when implementing the vast majority of our business solutions. However, we do have a few special cases where they don't fit—typically in highly configurable scenarios, such as when the name of the stored procedure to call is read from a configuration table and not known at compile time. Our team has deliberately chosen not to support such cases to avoid the additional framework complexity that would be introduced. Instead, we allow our developers to use the plain .NET APIs in such isolated cases.

The AAL solution itself isn't large and has been developed within a few man-months over a two-year period. Thus, our initial investment hasn't been very high and has already reached break-even status via lots of saved development and support hours.

Of course, the challenge of having a widely used and highly versatile platform is that it can become a single point of failure. We've mitigated this by having complete unit-test coverage of the AAL solution and by rolling out new versions of it in a governed service-by-service way. You could also argue that the AAL approach in itself introduces additional complexity into our system and forces our developers to learn a new abstraction layer. But we believe this is more than compensated for by the increased overall productivity and quality.

Another concern that we're aware of is that we must keep focus on overall design and architecture and not just do Windows services for everything simply because that approach has become so cheap and easy. Sometimes a third-party solution or an IIS-hosted Windows Process Activation Service (WAS) offers a better overall solution, even though it adds to the diversity of our production environment. ■

Ulrik Born *holds a Master of Science degree in Information Technology from the Technical University of Denmark and has been working for the past 15 years as developer and architect on the Windows platform. He's a lead developer for Saxo Bank, an Internet-based investment bank.*

Thanks to the following technical experts for reviewing this article: *Jonas Gudjonsson (Saxo Bank), James McCaffrey (Microsoft), Grigori Melnick (Microsoft) and Fernando Simonazzi (Microsoft)*

SpreadsheetGear

Performance Spreadsheet Components

SpreadsheetGear 2012 Now Available

NEW!

WPF and Silverlight controls, multithreaded recalc, 64 new Excel compatible functions, save to XPS, improved efficiency and performance, Windows 8 support, Windows Server 2012 support, Visual Studio 2012 support and more.

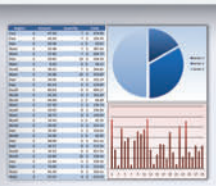
Excel Reporting for ASP.NET, WinForms, WPF and Silverlight

Easily create richly formatted Excel reports without Excel from any ASP.NET, Windows Forms, WPF or Silverlight application using spreadsheet technology built from the ground up for performance, scalability and reliability.

Excel Compatible Windows Forms, WPF and Silverlight Controls

Add powerful Excel compatible viewing, editing, formatting, calculating, filtering, charting, printing and more to your Windows Forms, WPF and Silverlight applications with the easy to use WorkbookView controls.

Excel Dashboards, Calculations, Charting and More



You and your users can design dashboards, reports, charts, and models in Excel or the SpreadsheetGear Workbook Designer rather than hard to learn developer tools and you can easily deploy them with one line of code.

Free 30 Day Trial

Download our fully functional 30-Day evaluation and bring Excel Reporting, Excel compatible charting, Excel compatible calculations and much more to your ASP.NET, Windows Forms, WPF, Silverlight and other Microsoft .NET Framework solutions.

www.SpreadsheetGear.com

Visual Studio Microsoft Partner

SpreadsheetGear

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

Implementing Static Code Analysis with StyleCop

Hamid Shahid

How often have you encountered code that's incomprehensible? Usually, it's inconsistent formatting, invalid comments and lack of naming conventions that make code unreadable. Such inconsistencies are often overlooked as petty anomalies, but they can make a big difference in the overall maintainability of code.

StyleCop is a great tool to maintain style and format consistency in your source code. Like Visual Studio Code Analysis, StyleCop performs static code analysis. However, unlike Visual Studio Code Analysis, it scans through source code files rather than managed code. Moreover, StyleCop only validates for style inconsistencies and doesn't perform code optimization and performance checks.

In this article, I'll introduce StyleCop, show you how it works and discuss what factors you should consider when adopting it in your project. Finally, I'll demonstrate how to include StyleCop execution in your Visual Studio Team Foundation Server (TFS) build.

This article discusses:

- The basics of StyleCop
- Adding StyleCop to a Team Build
- Using customized build templates

Technologies discussed:

StyleCop, Visual Studio Team Foundation Server

StyleCop Essentials

StyleCop (stylecop.codeplex.com) is an open source tool that performs static code analysis on C# source files. It's integrated with Visual Studio and appears in the context menu, giving you the option to scan the current file or any selected files or projects. **Figure 1** shows StyleCop options available on the context menu for a Visual Studio project.

When you select the Run StyleCop or the Run StyleCop (Rescan All) option, StyleCop parses all the C# files and validates them for the designated StyleCop rules. If there are violations, a warning appears in the Error List window.

The StyleCop Settings File This is where StyleCop keeps all its configuration options. This file contains information such as selected rules; vocabulary information such as custom words or acronyms; and whether to merge the settings file with the settings file in the parent directories, if there are any.

Given that StyleCop recursively looks for the settings file in the source file's parent directory, it's a best practice to keep a single version of the Settings.StyleCop file. Maintaining one file and storing it at the root of the team project ensures you use the same set of rules across the whole team project.

Custom Dictionary File In addition to allowing adding words and acronyms to the Settings file, StyleCop also works with a CustomDictionary.xml file that uses the same format as the Visual Studio Code

Analysis dictionary. This lets you use the same dictionary file for both. **Figure 2** displays the format of the dictionary file.

Having explained the purpose and basics of StyleCop, I'll now go through what's involved in making it an integral part of your development team's working practice.

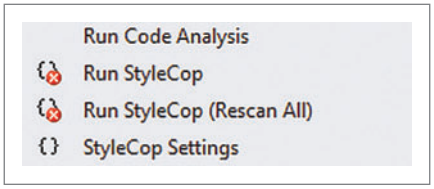


Figure 1 **StyleCop Context Menu Options**

StyleCop Rules This is a check that StyleCop performs on a code file. There are a number of rules available out of the box, and you have the option of writing your own custom rules if you wish. The StyleCop wiki details how to write your own StyleCop rules (see bit.ly/12P665L).

The first step in using StyleCop is deciding which StyleCop rules to use. I strongly advise using all the StyleCop rules. However, development teams often have their own coding standards, and there might be a strong push-back on the adoption of certain StyleCop rules. You have to balance the long-term benefits of consistently styled, maintainable code against the small inconvenience it might cause. Like so many good practices, once you get into the habit of using StyleCop, it becomes second nature. In any case, it's essential to agree on the StyleCop rules your team will use across the board.

StyleCop rules fall in the following seven categories:

1. **Documentation Rules:** Validate the suitability of documentation elements in the source files.
2. **Layout Rules:** Validate layout and line spacing in the source files.
3. **Maintainability Rules:** Validate the source files for maintainability aspects such as an unwanted parenthesis or the existence of multiple classes in a single file.
4. **Naming Rules:** Validate the substitutability of method and variable names.
5. **Ordering Rules:** Validate the code content is ordered correctly.
6. **Readability Rules:** Validate the code is properly formatted and readable.
7. **Spacing Rules:** Validate the spacing in the code content is valid and appropriate.

Figure 2 **The Custom Dictionary File**

```
<Dictionary>
  <Words>
    <Unrecognized>
      <Word/>
    </Unrecognized>
    <Recognized>
      <Word/>
    </Recognized>
    <Deprecated>
      <Term PreferredAlternate=""/>
    </Deprecated>
    <Compound>
      <Term CompoundAlternate=""/>
    </Compound>
    <DiscreteExceptions>
      <Term />
    </DiscreteExceptions>
  </Words>
  <Acronyms>
    <CastingException>
      <Acronym />
    </CastingException>
  </Acronyms>
</Dictionary>
```

You can read more about the StyleCop categories and their respective rules in the StyleCop rules documentation at bit.ly/191GgiQ.

Adding StyleCop to Your Team Build

It's all well and good to have StyleCop rules selected and stored in a settings file, but the only way to ensure StyleCop is executed consistently for all source code is to run it as part of the build process.

There are two ways to do this:

1. Integrate StyleCop with the C# project's MSBuild file so that it's executed whenever the project is compiled. The StyleCop documentation (bit.ly/13ZX2xL) describes how to do so.
2. Add StyleCop to your continuous integration Team Build so that it's executed for every check-in.

I'll explain how to run StyleCop in your continuous integration Team Build. If you're using gated builds, executing StyleCop will ensure any code files with violations aren't checked in. If not used in gated builds, the broken build will still prompt you to fix the violations when you check in code.

In order to run StyleCop in Team Build, the process must be invoked from an activity in the build workflow. I'll use the StyleCop activity from an open source project called Community TFS Build Extensions (tfsbuildextensions.codeplex.com). The Community TFS Build Extensions is a set of libraries containing a number of reusable workflow activities that you can simply drag and drop into your Team Build process template.

Build Controller Changes The first thing you need to do before customizing the Team Build is set the custom assemblies path of

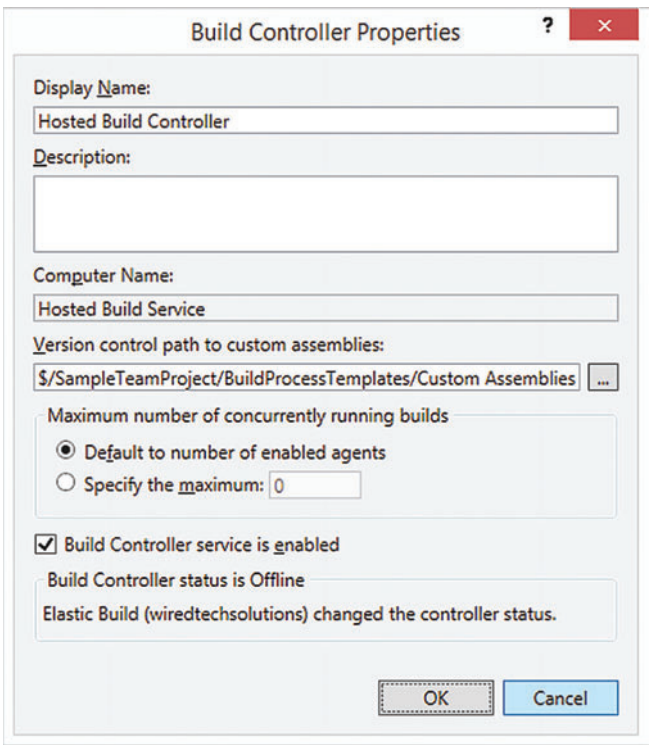


Figure 3 **Build Controller Properties**

your build controller. This is the location from where the build agents load assemblies for any custom activities they find in the build workflow.

To add custom assemblies, create a new folder at an appropriate location in your Team Project. I named the new folder Custom Assemblies and created it below the BuildProcessTemplate just below the Team Project root folder. Now, check in the following assemblies:

- StyleCop.dll
- StyleCop.CSharp.dll
- StyleCop.CSharp.Rules.dll
- TFSBuildExtensions.Activities.dll
- TFSBuildExtensions.Activities.StyleCop.dll

The next step is to configure your build controller to use these assemblies. To do so:

1. Click on the build link in Team Explorer. Click Actions and select Manage Build Controllers.
2. From the dialog box that appears, select your build controller and click on the Properties button.
3. In the Build Controller Properties dialog box, set the "Version control path to custom assemblies" property to the Custom Assemblies folder that was created earlier in the Team Project, as shown in **Figure 3**.

Click OK and close the properties dialog box. At this point, the build controller is configured to load your custom activities. The next step is to customize your build's template.

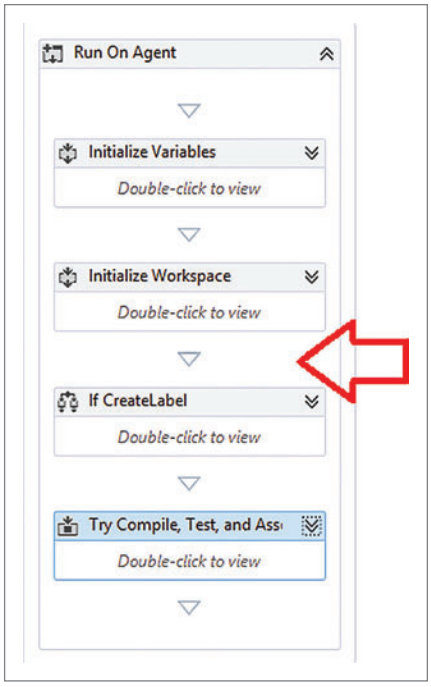


Figure 4 **StyleCop Activity Drop Location**

Custom Build Templates

For every new Team Project, TFS creates a number of build templates out of the box. These build templates are created in a folder called ProcessBuildTemplates that resides at the root of the Team Project. Start by taking a copy of the DefaultTemplate.11.1.xaml template and customizing it to add the StyleCop activity. I created a copy of the file DefaultTemplate.11.1.xaml and renamed it CustomTemplate.xaml.

To customize the build workflow, you'll need to add the custom activities to your development environment. To do so, create a new workflow activity library project in Visual Studio. In the Add New Project dialog, make sure the Microsoft .NET Framework 4.5 is selected as the target platform. The next step is to add a link to the file CustomTemplate.xaml in the newly created project. To do that, right-click on the project, select Add Existing Item, browse to the file CustomTemplate.xml and click the Add as Link button.

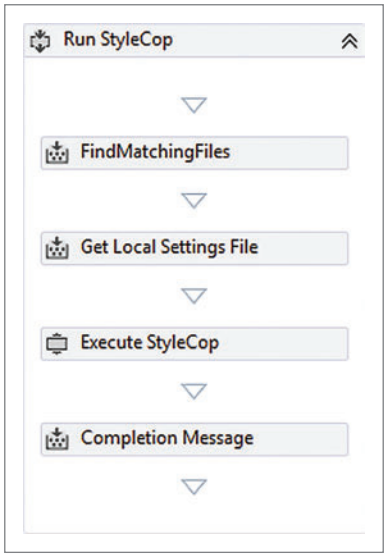


Figure 5 **The Run StyleCop Sequence**

The last step in setting up your development environment is to add the StyleCop activity in the Toolbox window for allowing drag-and-drop. To do so, right-click the area below "activities" in your Toolbox window and select the option Add Tab. Name the new tab "TFS Build Extensions." Right-click the tab name and select "Choose items." Browse to the assembly TfsBuildExtensions.Activities.Stylecop.dll and click OK. You can now open the CustomTemplate.xaml file and drag the StyleCop activity on to it.

Build Template Customization You should run StyleCop early in the build process. This allows the build to fail quickly if any violations are encountered. Because StyleCop requires the source files to scan, the first place where StyleCop can be executed is after the Initialize Workspace sequence within the Run On Agent sequence, as shown in **Figure 4**.

Having determined the appropriate location in the build workflow to add StyleCop activity, the next step is to add a sequence activity. Rename the sequence activity to Run StyleCop. The final listing of my Run StyleCop sequence is shown in **Figure 5**.

Code Walk-Through **Figure 6** details the variables defined in the Run StyleCop sequence, their types and their respective purposes.

The workflow also contains a parameter called StyleCopSettingsFile of type String that stores the path of the StyleCop settings file.

The first activity in the Run StyleCop sequence is the FindMatchingFiles activity. The activity is contained in the Microsoft.TeamFoundation.Build.Workflow.dll assembly and returns the list of all files matching the given file pattern. **Figure 7** describes how the properties of this activity are set.

The activity is passed the pattern of finding all C# (*.cs) files in the Build Directory, and it returns the result in the SourceCodeFiles enumeration.

The next activity in the sequence is the ConvertWorkspaceItem activity, which resides in the assembly Microsoft.TeamFoundation.Build.Workflow.Activities.dll. The activity converts the server path of the given StyleCop Settings file—passed as a parameter—to a local path on the build server. The properties of this activity are shown in **Figure 8**.

Now that the source file names are retrieved and the location of the StyleCop settings is established, the next activity in the Run StyleCop sequence is the StyleCop activity. **Figure 9** displays how the properties of the StyleCop activity are set.

The activity takes the enumeration SourceCodeFiles—converted to an array—as input and returns the result and violation counts in

Figure 6 **Variables Defined in Run StyleCop Sequence**

Variable Name	Type	Description
SourceCodeFiles	IEnumerable<String>	Stores the names of all files to be scanned by StyleCop.
IsSuccess	Boolean	Stores whether the StyleCop activity found any violations.
ViolationCount	Int32	Stores the count of StyleCop violations.

Figure 7 **FindMatchingFiles Activity Properties**

Property Name	Value
DisplayName	FindMatchingFiles
IsSuccess	String.Format("{0}***.cs", BuildDirectory)
Result	SourceCodeFiles

the IsSuccess and ViolationCount variables, respectively. It's given the display name Execute StyleCop and is set to treat warnings as errors and to fail the build if any errors are encountered.

The final activity in the Run StyleCop sequence is the Write Build Message activity. The activity is set to display the result and violations count. **Figure 10** displays how the properties of this activity are set.

Your Custom build template is now ready for use. Save and check in the file CustomTemplate.xaml. To use the new build template, open your build definition, click process, expand Build Process Template and click the New button. In the New Process Template dialog box that's shown, select the option "Select an existing XAML file" and browse to the CustomTemplate.xaml file.

Figure 8 **Get Local Settings File Properties**

Property Name	Value
Direction	ServerToLocal
DisplayName	Get Local Settings File
Input	StyleCopSettingsFile
Result	StyleCopSettingsFileLocalPath
Workspace	Workspace

Figure 9 **Execute StyleCop Properties**

Property Name	Value
DisplayName	Execute StyleCop
LogExceptionStack	True
SettingsFile	StyleCopSettingsFile
ShowOutput	True
SourceFiles	SourceCodeFiles.ToArray()
Result	StyleCopSettingsFileLocalPath
Succeeded	IsSuccess
TreatWarningsAsErrors	True

Figure 10 **Write Build Message Activity Properties**

Property Name	Value
DisplayName	Completion Message
Importance	Microsoft.TeamFoundation.Build.Client.BuildMessageImportance.Normal
Message	String.Format("StyleCop was completed with {0} violations", StyleCopViolationsCount)

StyleCop Considerations

Here are some considerations of which to be aware:

- Unlike Visual Studio Code Analysis, StyleCop doesn't support Visual Basic .NET and can only be used for source files written in C#.
- At the time of writing this article, StyleCop wasn't yet available for Visual Studio 2013.
- The Hosted Build Controller is a build controller hosted in the cloud by Visual Studio Team Foundation Service. The steps to configure this build controller are the same if you're using an on-premises build server.
- This article used Team Foundation Server (TFS) 2012. The steps are the same for TFS 2010 and TFS 2013. Make sure you're using the correct version of TFS Build Extensions.

Set the value of parameter StyleCopSettingsFile to the location of your Settings.StyleCop file. Click Save to save the build definition. Your build with StyleCop is now ready for use. It's best to use this build template for your gated builds. This will ensure none of the source files checked in have any StyleCop violations.

> Static code analysis promotes better coding standards, and it can run in your Team Build to ensure it adheres to your standards.

Next Steps

I've demonstrated how you can use StyleCop to enforce static code analysis in your Team Build. Static code analysis promotes better coding standards, and it can run in your Team Build to ensure all checked-in code adheres to your standards. You can similarly enforce other best practices in your Team Build. Microsoft ALM Rangers have produced a number of useful build templates that you can use in the Team Foundation Build Customization Guide (vsarbuildguide.codeplex.com) project. Moreover, you have the option of writing your own activities or using the activities available in the Community TFS Build Extensions project. ■

Hamid Shahid *is a Microsoft ALM Ranger and an independent consultant with more than 12 years of experience designing and developing enterprise software. He has keen interest in promoting best practices in Microsoft ALM technologies. He can be reached via his blog at hamidshahid.blogspot.com and can be followed on Twitter at twitter.com/hamid_shahid.*

THANKS to the following ALM Rangers and technical experts for reviewing this article: *Mike Fourie (independent consultant), Willy-Peter Schaub (Microsoft) and Patricia Wagner (Microsoft)*

JAMES MCCAFFREY

Radial Basis Function Networks for Programmers

Radial basis function (RBF) networks are software systems that have certain similarities to neural networks. An RBF network accepts one or more numeric input values, such as (1.0, -2.0, 3.0), and generates one or more numeric output values, such as (4.6535, 9.4926). RBF networks (sometimes called radial nets) can be used to classify data and make predictions. For example, an RBF network could be used to predict the scores of two football teams that are scheduled to play each other, based on historical data such as each team's current winning percentage, home field advantage (-1.0 or +1.0) and so on. Or an RBF network could be used to classify a hospital patient's risk of cancer (low = 0, high = 1) based on the values of medical test results and other factors such as age and sex.

In my opinion, RBF networks are among the most fascinating of all machine-learning techniques. But even though there are many research papers that explain the complex mathematics behind these networks, there are very few resources that explain them from a programmer's point of view. This article will describe exactly what RBF networks are, explain how they compute their outputs and illustrate with a complete RBF network input-output demo program.

The best way for you to see where this article is headed is to take a look at the demo program in **Figure 1**. The demo creates an RBF network, configures it, sends an input vector with values of (1.0, -2.0, 3.0), and displays the output vector of (4.6535, 9.4926). The demo begins by creating a 3-4-2 RBF network. The 3 indicates an input to the radial net has three numeric values. The 2 indicates there are two numeric outputs. The 4 indicates the radial net has four hidden, internal processing nodes.

```
file:///C:/RadialNetworksInputOutput/bin/Debug/RadialNetworksInputOutput.EXE
Begin Radial Basis Function (RBF) network input-output demo

Creating a 3-4-2 radial net

Setting centroids (means) to:
-3.00 -3.50 -3.80
-1.00 -1.50 -1.80
 2.00  2.50  2.80
 4.00  4.50  4.80

Loading centroids into radial net

Setting standard deviations (widths) to:
 2.22  3.33  4.44  5.55

Loading standard deviations into radial net

Setting hidden-output weights to:
 5.00 -5.10
-5.20  5.30
-5.40  5.50
 5.60 -5.70

Loading hidden-output weights into radial net

Setting output biases to:
 7.0  7.1

Loading output biases into radial net

Setting x-input to:
 1.0 -2.0  3.0

Computing the output of the radial net

Hidden[0] distance = 8.0306
Hidden[0] output = 0.0014

Hidden[1] distance = 5.2240
Hidden[1] output = 0.2921

Hidden[2] distance = 4.6141
Hidden[2] output = 0.5828

Hidden[3] distance = 7.3817
Hidden[3] output = 0.4129

The output of the RBF network is:
 4.6535  9.4926

End RBF network demo
```

Figure 1 **Radial Basis Function Network Input-Output Demo**

The radial net requires four sets of configuration information, usually referred to as the parameters of the system. The first parameter is a set of so-called centroids. Centroids are sometimes called means. In the demo, the centroids are (-3.0, -3.5, -3.8), (-1.0, -1.5, 1.8), (2.0, 2.5, 2.8) and (4.0, 4.5, 4.8). Notice there's one centroid for every hidden node and that each centroid has the same number of values as an input vector. Because the purpose of the demo is only to explain how RBF networks compute their output, rather than to solve a realistic problem, the number of hidden nodes (four) is artificially small, and the values of the four centroids are arbitrary.

The second set of parameters for the demo radial net is four standard deviations. Standard deviations are sometimes called RBF widths. The values in the demo are 2.22, 3.33, 4.44 and 5.55. There's one standard deviation for every hidden-node processing unit. Again, the values used in the demo are dummy values and don't correspond to a real problem.

The third set of RBF network parameters is the weights. Notice in **Figure 1** that there are eight weights. If an RBF net has j hidden

Code download available at archive.msdn.microsoft.com/mag201310TestRun.

nodes and k output nodes, there will be j * k weights. Here, the 4 * 2 = 8 weights are 5.0, -5.1, -5.2, 5.3, -5.4, 5.5, 5.6 and -5.7. Like the other sets of parameters, these values are purely arbitrary and are for demonstration purposes only.

The fourth set of radial net parameters in the demo is two bias values. The number of bias values in an RBF network is equal to the number of output values, two in this case. The two dummy bias values are 7.0 and 7.1.

After loading the four sets of parameter values into the RBF network, an input of (1.0, -2.0, 3.0) is fed to the network. Using the parameter values, an output of (4.6535, 9.4926) is computed. The demo program displays some intermediate values of the computation—a distance and output for each hidden node—but these intermediate values are shown only to help you understand the RBF input-output mechanism and aren't normally displayed.

This article assumes you have at least intermediate-level programming skills with a C-family language, but doesn't assume you know anything about radial basis function networks. The demo program is coded using C#, but you should have no trouble refactoring my code to another language such as Python or Windows PowerShell. I've removed most normal error checking to keep the main ideas clear. All the key code for the demo program is presented in this article, but I've omitted some of the helper display methods. The complete source code for the demo is available at archive.msdn.microsoft.com/mag201310TestRun.

The RBF Input-Output Mechanism

In order to understand the RBF input-output mechanism, take a look at the diagram in **Figure 2**. The values in the diagram correspond to the ones in the demo program. Processing occurs in two major steps. First, an input vector is sent to each hidden node and each hidden node independently computes an intermediate output value. Second, the intermediate hidden-node output values are combined to produce the final system output values.

The diagram in **Figure 2** uses zero-based indexing for all objects. Expressed mathematically, the output of a hidden node j is:

$$\phi_j(x) = e^{-\frac{\|x-\mu_j\|^2}{2\sigma_j^2}}$$

This equation is an example of what's called the Gaussian function and when graphed has a characteristic bell-shaped curve. The bell-shaped icon at the top of each hidden node indicates that the hidden-node outputs are computed using a Gaussian function.

The equation is simpler than it may initially appear. The Greek letter phi represents the output of a hidden node, j in this case. The x represents the input. Here x is a vector (1.0, -2.0, 3.0) rather than a single scalar value. Lowercase e is Euler's constant (2.71828 ...) and e raised to some value corresponds to the Exp function available in most programming languages. The Greek letter mu is the j^{th} centroid vector. The pair of double-bar symbols, when applied to the difference between two vectors, is equivalent to a shorthand notation for the Euclidean geometry distance function, which is the square root of the sum of the squared differences between the components of the two vectors. The Greek letter sigma represents the standard deviation of the j^{th} hidden node.

I'll demonstrate how the intermediate output of the bottommost hidden node [3] is computed using the values of the demo program. The input x is (1.0, -2.0, 3.0). The centroid, mu, is (4.0, 4.5, 4.8). The standard deviation, sigma, is 5.55. Recall that these values are purely arbitrary and for demonstration purposes, and do not correspond to a real problem.

The distance between x and mu is Sqrt((1.0 - 4.0)^2 + (-2.0 - 4.5)^2 + (3.0 - 4.8)^2) = Sqrt(9.00 + 42.25 + 3.24) = 7.3817 as shown in **Figure 1**. The distance squared is 54.49. Notice that the squaring operation undoes the square root operation of the distance formula, which means that the equation could've been simplified a bit.

Putting all the values together, the output for the hidden node 3 is Exp(-54.49 / (2 * (5.55)^2)) = Exp(-0.88451) = 0.4129, which is the value shown in **Figure 1**.

The output values for hidden nodes 0, 1 and 2 are calculated in the same way and are 0.0014, 0.2921 and 0.5828, respectively.

Now I'll show how the hidden-node outputs are combined to yield the final RBF network outputs. An RBF network's output is the weighted sum of products of all hidden-node outputs times an associated weight, plus a final bias value. Expressed as an equation, the value of output node k is:

$$y_k(x) = \left(\sum_{j=0}^{M-1} w_{jk} * \phi_j(x) \right) + b_k$$

Here, k is the index of an output node (0, 1 in the demo); j is the index of a hidden node (0, 1, 2, 3); and M is the number of hidden nodes. The term w_{jk} is the weight from hidden node j to output node k. The term b_k is the bias value associated with output k.

For example, if h0, h1, h2, and h3 (using the easier-to-type letter h instead of Greek letter phi) are the outputs of the hidden nodes 0 through 3, then final output 0 is computed as (w00 * h0) + (w10 * h1) + (w20 * h2) + (w30 * h3) + b0 = (5.0 * 0.0014) + (-5.2 * 0.2921) + (-5.4 * 0.5828) + (5.6 * 0.4129) + 7.0 = 0.0070 + -1.5189 + -3.1471 + 2.3122 + 7.0 = 4.6535 (rounded), as shown in **Figure 1**.

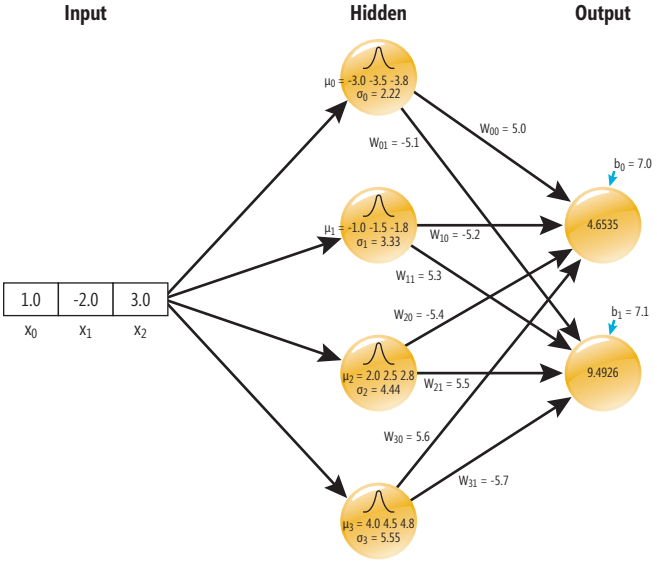


Figure 2 **Radial Basis Function Network Architecture**

Demo Program Structure

The overall structure of the demo program, with most WriteLine statements removed and some minor edits, is presented in **Figure 3**. The RBF network functionality is contained in class RadialNet. Class Helpers contains three display methods.

To create the demo program, I launched Visual Studio 2012. The demo has no significant .NET dependencies, so any version of Visual Studio should work. I created a new C# console application project named RadialNetworksInputOutput. After the template code loaded, I renamed file Program.cs to RadialNetIOProgram.cs and Visual Studio automatically renamed class Program accordingly. I deleted all unnecessary using statements at the top of the source code.

You should be able to understand the statements in the Main method without difficulty. The statement that instantiates the RBF network object is:

```
RadialNet rn = new RadialNet(numInput, numHidden, numOutput);
```

Figure 3 **RBF Network Demo Program Structure**

```
using System;
namespace RadialNetworksInputOutput
{
  class RadialNetIOProgram
  {
    static void Main(string[] args)
    {
      Console.WriteLine("\nBegin Radial Basis Function (RBF) network demo\n");

      int numInput = 3;
      int numHidden = 4;
      int numOutput = 2;

      Console.WriteLine("\nCreating a 3-4-2 radial net");
      RadialNet rn = new RadialNet(numInput, numHidden, numOutput);

      double[][] centroids = new double[4][];
      centroids[0] = new double[] { -3.0, -3.5, -3.8 };
      centroids[1] = new double[] { -1.0, -1.5, -1.8 };
      centroids[2] = new double[] { 2.0, 2.5, 2.8 };
      centroids[3] = new double[] { 4.0, 4.5, 4.8 };
      rn.SetCentroids(centroids);

      double[] stdDevs = new double[4] { 2.22, 3.33, 4.44, 5.55 };
      rn.SetStdDevs(stdDevs);

      double[][] hoWeights = new double[4][];
      hoWeights[0] = new double[2] { 5.0, -5.1 };
      hoWeights[1] = new double[2] { -5.2, 5.3 };
      hoWeights[2] = new double[2] { -5.4, 5.5 };
      hoWeights[3] = new double[2] { 5.6, -5.7 };
      rn.SetWeights(hoWeights);

      double[] oBiases = new double[2] { 7.0, 7.1 };
      rn.SetBiases(oBiases);

      double[] xValues = new double[3] { 1.0, -2.0, 3.0 };
      Console.WriteLine("\nSetting x-input to:");
      Helpers.ShowVector(xValues, 1, 4, true);

      Console.WriteLine("\nComputing the output of the radial net\n");
      double[] yValues = rn.ComputeOutputs(xValues);

      Console.WriteLine("\nThe output of the RBF network is:");
      Helpers.ShowVector(yValues, 4, 4, true);

      Console.WriteLine("\nEnd RBF network demo\n");
      Console.ReadLine();
    } // Main
  } // Program

  public class RadialNet { ... }
  public class Helpers { ... }

} // ns
```

The four statements that load RBF network parameter values are:

```
rn.SetCentroids(centroids);
rn.SetStdDevs(stdDevs);
rn.SetWeights(hoWeights);
rn.SetBiases(oBiases);
```

The statement that loads the input and computes and returns the RBF network output is:

```
double[] yValues = rn.ComputeOutputs(xValues);
```

The RadialNet Class

The definition of the RBF network class begins with:

```
public class RadialNet
{
  private int numInput;
  private int numHidden;
  private int numOutput;

  private double[] inputs;
  private double[][] centroids; // Aka means
  private double[] stdDevs; // Aka widths
  private double[][] hoWeights;
  private double[] oBiases;
  private double[] outputs;
...
```

The purpose of each of these class members should be clear, based on the explanation of how an RBF network computes its output. The weights are stored as an array-of-arrays-style matrix where the first index indicates the hidden node and the second index indicates the output node. The C# language, unlike most languages, supports a true matrix data type, and you may wish to use it instead of an array of arrays for the weights matrix.

The RadialNet constructor is defined as:

```
public RadialNet(int numInput, int numHidden, int numOutput)
{
  this.numInput = numInput;
  this.numHidden = numHidden;
  this.numOutput = numOutput;

  this.inputs = new double[numInput];
  this.centroids = MakeMatrix(numHidden, numInput);
  this.stdDevs = new double[numHidden];
  this.hoWeights = MakeMatrix(numHidden, numOutput);
  this.oBiases = new double[numOutput];
  this.outputs = new double[numOutput];
}
```

If you refer to the diagram in **Figure 2**, you'll see how the size of each class array and matrix is related to numInput, numHidden and numOutput. A static utility method, MakeMatrix, is called by the constructor just to keep the code a bit cleaner. Method MakeMatrix is defined as:

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

Class RadialNet has four methods to set the values of the centroids, standard deviations, weights and biases. An alternative design is to overload the constructor to accept all RBF parameter values, but I prefer separate set methods in most situations. Method SetCentroids is:

```
public void SetCentroids(double[][] centroids)
{
  if (centroids.Length != numHidden)
    throw new Exception("Bad number of centroids");
  if (centroids[0].Length != numInput)
    throw new Exception("Bad centroid size");

  for (int i = 0; i < numHidden; ++i)
    for (int j = 0; j < numInput; ++j)
      this.centroids[i][j] = centroids[i][j];
}
```

Method SetStdDevs is:

```
public void SetStdDevs(double[] stdDevs)
{
  if (stdDevs.Length != numHidden)
    throw new Exception("Bad number of stdDevs");
  Array.Copy(stdDevs, this.stdDevs, stdDevs.Length);
}
```

Method SetWeights is:

```
public void SetWeights(double[][] hoWeights)
{
  if (hoWeights.Length != numHidden)
    throw new Exception("Bad number of weights");
  if (hoWeights[0].Length != numOutput)
    throw new Exception("Bad number of weights");
  for (int i = 0; i < numHidden; ++i)
    for (int j = 0; j < numOutput; ++j)
      this.hoWeights[i][j] = hoWeights[i][j];
}
```

Method SetBiases is:

```
public void SetBiases(double[] oBiases)
{
  if (oBiases.Length != numOutput)
    throw new Exception("Bad number of hoBiases");
  Array.Copy(oBiases, this.oBiases, oBiases.Length);
}
```

The ComputeOutputs Method

The heart of the RadialNet class is method ComputeOutputs. The method's definition begins:

```
public double[] ComputeOutputs(double[] xValues)
{
  Array.Copy(xValues, inputs, xValues.Length);
  double[] hOutputs = new double[numHidden];
...
```

Here, x-input values are simply copied into member array inputs. Because method ComputeOutputs uses but doesn't change the x-input values, member array inputs isn't really needed. However, in some cases you might want to perform some preliminary processing of the x-input. And I feel that using an explicit class inputs array is a cleaner design and worth the extra work of copying in values. The local hOutputs array holds the outputs of each hidden node. An alternative design is to define hOutputs as a class member.

Next, ComputeOutputs computes the outputs for each hidden node:

```
for (int j = 0; j < numHidden; ++j)
{
  double d = Distance(inputs, centroids[j]);
  // Display distance here if you wish
  double r = -1.0 * (d * d) / (2 * stdDevs[j] * stdDevs[j]);
  double g = Math.Exp(r);
  // Display hidden output here if you wish
  hOutputs[j] = g;
}
...
```

The distance is computed by a static utility method, Distance, which is defined:

```
public static double Distance(double[] x, double[] c)
{
  double sum = 0.0;
  for (int i = 0; i < x.Length; ++i)
    sum += (x[i] - c[i]) * (x[i] - c[i]);
  return Math.Sqrt(sum);
}
```

Notice that Distance performs a square root operation and that this value is stored into variable d, which is then squared. An alternative is to define a method DistanceSquared, which is the same as Distance except for the call to the square root function, and then not square the value of d. Although this approach is more efficient, it makes the code out of sync with the standard math definitions used in RBF literature.

The code in ComputeOutputs that calculates the final outputs is:

```
for (int k = 0; k < numOutput; ++k)
  outputs[k] = 0.0;

for (int k = 0; k < numOutput; ++k)
  for (int j = 0; j < numHidden; ++j)
    outputs[k] += (hOutputs[j] * hoWeights[j][k]);

for (int k = 0; k < numOutput; ++k)
  outputs[k] += oBiases[k];
...
```

Method ComputeOutputs finishes up by copying the values in the member outputs array to a return value:

```
...
  double[] result = new double[numOutput];
  Array.Copy(outputs, result, outputs.Length);
  return result;
}
```

An alternative design is to define ComputeOutputs using a void return type, and define a separate GetOutputs method.

Wrapping Up

The code and explanation presented in this article should get you started if you want to explore radial basis function networks. There are many different types of RBF networks. The RBF net in this article uses a Gaussian function to compute hidden-node output. RBF networks can use many other functions, with names such as multi-quadratic and thin-plate spline. The particular function used by an RBF network is called the kernel of the network.

You may be wondering exactly where the rather exotic architecture of RBF networks comes from. RBF networks were a result of academic research into the theory of function approximation. It can be proved that, with a few assumptions and loosely speaking, RBF networks can replicate any mathematical function. This means in theory at least, RBF networks can be used to make predictions on data that follows any underlying model. Neural networks share this universal-function characteristic. In my opinion, it's not clear whether RBF networks are more or less effective than neural networks, or roughly the same, when it comes to machine-learning classification and prediction. However, there's good evidence that RBF networks can be trained much more quickly than neural networks, and unlike neural networks, which typically require very large amounts of training data, RBF networks can work well even with small amounts of training data.

In order to use an RBF network to make predictions for a realistic problem, the network must be trained. This means using existing historical data and finding the best set of centroids, standard deviations, weights and bias values—that is, the set of parameter values that generates RBF network outputs that most closely match the historical data. Training an RBF network also involves finding the optimal number of hidden nodes. Training RBF networks is an interesting problem that will be explained in a future article. ■

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

Thanks to the following technical expert for reviewing this article: *Dan Liebling (Microsoft Research)*

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

TED NEWARD

Getting Started with Oak

It's interesting, sometimes, to look at the changes that have occurred within our industry over the past 50 years. A college graduate in 1960 could, quite reasonably, have gotten a job as a programmer and, across the breadth of her career, watched her job go from filling out punch cards to programming a machine the size of a room with a whopping 4K of active memory (and no online storage); seen the rise of programming languages such as COBOL on mainframes, and then minicomputers such as the IBM PC; experienced changes in UI technology such as the GUI; watched the laptop become the standard computer of choice; witnessed the "birth" of the Internet and the Web—and eventually retire right about the time that tablets and mobile devices are becoming the consumer computing devices of choice.

But it's also equally true that changes occur within certain pockets of the industry, and the same kinds of "revolutions" take place within a smaller circle of technology or domain. One such swiftly changing circle of technology is that of the ubiquitous Web site: from Common Gateway Interface (CGI) scripts written in Perl or C; to Web server extensions such as ISAPI; to "page-oriented" tools such as ASP; to more powerful underpinnings behind those page-oriented tools (ASP.NET, backed by the CLR instead of native code); and so on. Increasingly, the approach of choice for these tools is moving away from "statically typed" languages, such as C#, to more "dynamically typed" languages, such as Ruby or Python. Debates rage, however, over the severity of penalties to programming with a dynamic language (wherein the compiler can't catch common mistakes) and how they trade off against the overhead (the compilation process and having to work with or around the type system) of a static language. These debates have yet to yield an overwhelming "winner," and as a result, some developers are starting to look at ways to take the best of both worlds.

Oak, an open source project from Amir Rajan (Twitter: @amirrajan; Web: amirrajan.net), is one such attempt to leverage the dynamic capabilities inherent within C# (via the "dynamic" keyword/type and the Dynamic Language Runtime, or DLR, both introduced in Visual Studio 2010) and gain the productivity benefits of a system such as Ruby on Rails, but keep it all in a language (C#) that offers static type checking and verification.

> Oak is more of a development lifestyle approach than "just" a library.

Figure 1 **List of Supported Rakefile Targets from the Command Line**

```
C:\Projects\Exploration\Oak\Blog>rake -T
rake build                  # builds the solution
rake default                # builds and deploys website to directories iis...
rake deploy                 # deploys MVC app to directory that iis express...
rake export                 # if you have the nuget package oak installed, ...
rake reset                  # if you have the nuget package oak installed, ...
rake sample                 # if you have the nuget package oak installed, ...
rake server                 # start iis express for MVC app
rake simulate_load_balance  # simulate the web application as if it were lo...
rake stop_nginx             # stops nginx
rake sync[file]             # synchronizes a file specfied to the website d...
rake tests                  # run nspec tests
rake ui                     # run ui automation tests
rake ui_tests               # runs ui tests (without building)
```

Setting Up

Oak is available as a NuGet package, but one thing that developers coming to Oak must understand is that, before all other things, Oak is more of a development lifestyle approach than "just" a library. Oak will take a different approach to building applications from what the traditional ASP.NET MVC developer may be used to. For example, emphasis on extremely rapid feedback will mean Oak wants to be continuously building in the background (or at least appear to be), such that simply saving the .cs file will be enough to view the results in the browser—no explicit "build" step should be necessary. (How Oak accomplishes this actually isn't rocket science, as you'll see.) The end result of Oak development will be a "traditional" ASP.NET application, so the operations guys will never know you built it differently from how you used to, fortunately.

Tooling

Part of this lifestyle adjustment is that Oak isn't just a NuGet download, but also requires the use of some other tools, which must be installed on your machine in order to get started. IIS Express is one such tool, which many Microsoft .NET Framework developers will already have if they've experimented with Visual Studio LightSwitch at all. Another such tool is Ruby, which is used in some of the build automation and tooling. Again, Ruby won't be needed on the production servers once development is done, but you'll need it to get started and to develop in Oak. As of this

writing, Ruby just released version 2.0, but the 1.9 release works just as well (and, for my money, gives me more comfort that I won't run into a weird 2.0 bug that nobody else has ever seen). Finally, as part of its rapid, interactive style, Oak wants to be able to flash messages at you from the continuous compilation system in the background, so Oak wants Growl for Windows, a notification system that will pop little messages to you onto the desktop (such as MSN Messenger used to) when events happen—such as a successful or unsuccessful build in the background. (I'll explore the Growl API in a later column, by the way, because it's a useful tool to have in your back pocket.)

> Ruby won't be needed on the production servers once development is done, but you'll need it to get started and to develop in Oak.

Once those are installed, there's one more step Oak requires: The Ruby scripts it uses need some packages available through the Ruby package manager, gem. (Ruby gem was, by the way, much of the inspiration for NuGet and other language package managers such as Node npm and Haskell Cabal.) Fire up a command prompt with Ruby on the PATH, and fire off this command from the command line:

```
C:\Projects\Exploration\Oak\Blog>gem install warmup
```

The warmup gem (bit.ly/15e51Lx) is actually how you'll get an Oak project started—it will pull down a template containing all the major moving parts Oak requires.

You can tell already that this is a long way away from the "File | New | Project" with which you're probably familiar.

Starting Oak

Getting a new Oak project up and running consists of using the warmup gem from the command line to pull down the seed template from its home online, like so:

```
C:\Projects\Exploration\Oak\Blog>warmup http://github.com/amirrajan/loam Blog
```

This seed project, once it's pulled down to your machine, contains a bare-bones "weblog" implementation (and I put weblog in quotes because if this can be called a weblog, then my NBA basketball aspirations are alive and well, despite the fact that I can't dribble, shoot or dunk). Once the disk stops spinning, open the downloaded Blog.sln Visual Studio Solution file (so you have the comfortable Visual Studio environment in front of you). Then, from the command line in which you issued the warmup command, run these two commands:

```
C:\Projects\Exploration\Oak\Blog>rake
C:\Projects\Exploration\Oak\Blog>rake server
```

Rake is the Ruby "build" tool, in much the same way that "make" was the C/C++ build tool for so many years; it will issue a sequence of steps according to a target. The first command will build the solution, and the second will tell Rake to deploy the code to the IIS Express directory and start IIS Express. Port 3000 (by default) will now be ready to accept incoming HTTP requests.

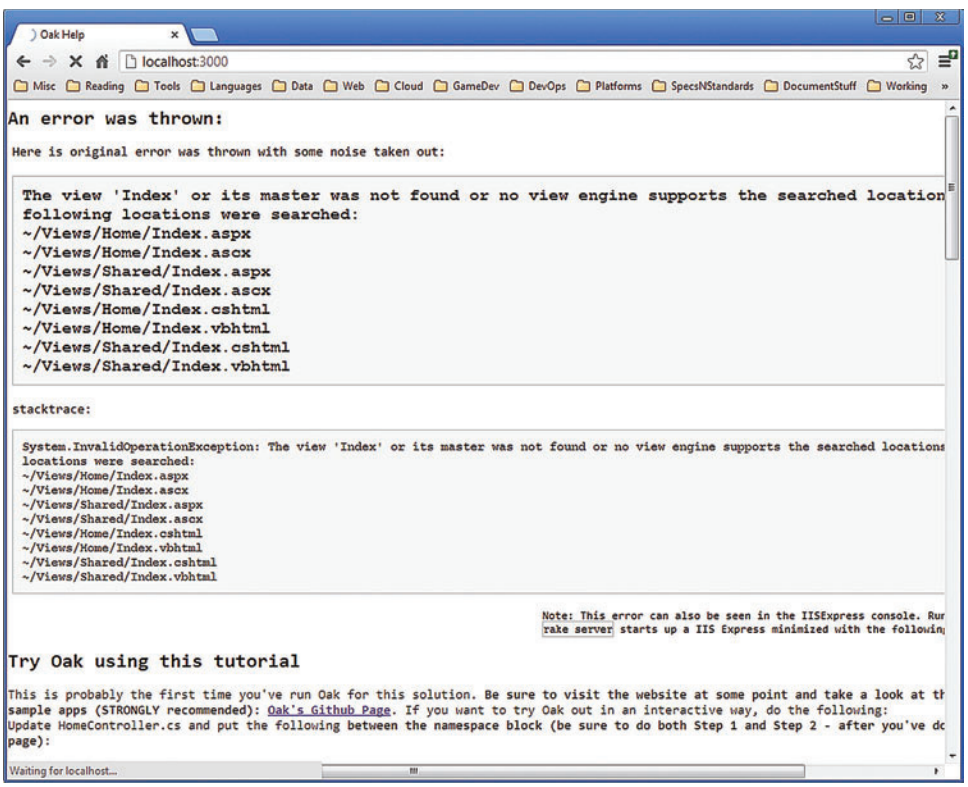


Figure 2 **Browsing to localhost:3000 to Ensure Oak Is Running**

If you're curious to see the complete list of targets that the Rakefile supports, call "rake -T," and you'll see the command-line output shown in **Figure 1**. (Note: A much longer, complete listing of targets is available with "rake -D," for those who want to see the full help text for each task.)

Once IIS Express is up and running, kick off the continuous-build tool, sidekick:

```
C:\Projects\Exploration\Oak\Blog>sidekick
```

Sidekick is actually a batch file that uses the command-line C# compiler to compile a small .exe program (sidekickapp.exe) that will set up a file-system watcher to see when files in the directory tree change, and—if they are source files—trigger an MSBuild to recompile the project. (Glenn

Block's recent experiments with Roslyn to create a C# REPL environment, ScriptCS, could also be used to accomplish the same thing, most likely.)

A quick aside before I go further: Notice in **Figure 1** how Rake offers an option to run "rake simulate_load_balance." This is—exactly as its name implies—a simulation of how the application will behave behind a simple round-robin load balancer, so you can test your application (and, specifically, how the application state will behave) in a load-balanced environment without having to fiddle around with virtual machines (VMs), load-balancer hardware or software, and IIS. It's a nifty little add-on to the whole project that almost justifies learning Oak entirely by itself.

At this point, Oak is up and running—and you can open a browser to localhost:3000 to prove it, as shown in **Figure 2**.

Don't stress about what Oak is telling you right now—that will be what I start exploring next. For now, it's enough to ensure that Oak is up and running.

Troubleshooting

If that doesn't work, it might be that Rake doesn't know where your IIS Express is expecting Web sites to reside; the Rakefile that Oak sets up uses a configuration file, dev.yml, to know where to install the ASP.NET-compiled files. In that file (in the same directory as the Blog.sln file), set the following lines to match what's on your system:

```
website_port: 3000
website_deploy_directory: c:\Prg\iisexpress\Blog
solution_name: Blog # just the name, no .sln extension
mvc_project: Blog # just the folder name,
  no .csproj extension
test_project: Blog.Tests # just the folder name,
  no .csproj extension
iis_express: C:\Program Files\IIS Express
```

On 64-bit systems, for example, IIS Express will be installed in "C:\Program Files (x86)\IIS Express."

More to Explore

There's a great deal of Oak yet to explore—in fact, the title of this article probably should've been "Acorn," given that this is the necessary seed from which the rest of the framework (and article series) will be born. Oak's got a lot yet to show you, starting with its integration into traditional ASP.NET MVC development structure (controllers, views and models), and carrying through into its database interactions. Plus, you have yet to see what all this additional tooling (Ruby and Growl) buys you. Hang out for a while longer—the real fun begins next time.

Happy coding! ■

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

Thanks to the following technical expert for reviewing this article: *Amir Rajan (Oak project creator)*

RACHEL APPEL

Build a Responsive and Modern UI with CSS for WinJS Apps

Working with CSS leaves many developers frustrated because it's full of nuances. Even minor changes to one selector or HTML element often propagate and affect another. Trying to keep CSS clean and effective can be quite difficult, considering its numerable quirks—especially when targeting multiple browsers. Luckily, if you're developing Windows Library for JavaScript (WinJS) apps, the CSS built in to the Visual Studio project templates is flexible yet consistent and maintainable. The Visual Studio project templates contain code that's the foundation of a responsive and modern UI design.

Built-in CSS for Windows Store Apps

All project templates for Windows Store apps built with JavaScript contain two basic style sheets or themes—dark (default) and light—in files named ui-dark.css and ui-light.css, respectively. These files are part of a core set of project files in the CSS folder under the Visual Studio project references node. There are nearly 4,000 lines of CSS in the core CSS files because they construct a UI that follows all the principles of Windows UI design, including responding to Windows snap view and media type changes. You shouldn't try to modify the built-in style sheets (they're read-only), but instead overwrite the built-in styles in your own CSS files.

To use the built-in Windows Store app CSS, add the following reference to the <head> section of any page (note that "2.0" in the link is for Windows 8.1, while "1.0" would be used for Windows 8):

```
<link href="//Microsoft.WinJS.2.0/css/ui-dark.css" rel="stylesheet" />
```

Two forward slashes at the start of the path indicate a reference to a WinJS core library (also referred to as "shared libraries," as opposed to local project files in your app package). Paths in Windows Store apps built with JavaScript that begin with a single forward slash denote the path starts at the project's root folder, and single forward slashes refer to local project files.

Windows Store apps built with JavaScript employ ample vendor prefixes (for example, -ms-grid, -flexbox and so on) so that the most technologically advanced CSS is available to use in apps. Other high-tech CSS you'll find in a WinJS app are CSS3 RGBA colors, Web Open Font Format (WOFF) fonts and media queries.

Style WinJS Controls with CSS

WinJS controls are extensions of HTML elements, so you style WinJS controls with the CSS just as you would any HTML tag. The Visual Studio project templates contain references to the CSS that creates modern UI styles for every HTML element and WinJS control, in both dark and light flavors. Of course, you aren't bound to these styles and are free to modify them as you see fit to complement your company or product branding or theme. However, it's important to maintain a consistency with the new Windows UI so as not to confuse users.

Figure 1 demonstrates the code for several basic and common HTML and WinJS controls using both the dark and light themes, respectively. **Figure 2** shows the output of this code.

As you can see from examining **Figure 2**, the default styles for controls reflect the new Windows UI. A complete listing of HTML and WinJS controls is available on the Windows Dev Center at bit.ly/w1jLM5.

Figure 1 **The Code for Basic HTML and WinJS Controls**

```
<div id="grid">
  <div style="-ms-grid-column: 1; -ms-grid-row: 1">
    <label for="textbox">Textbox:</label>
    <input id="textbox" type="text" />
  </div>
  <div style="-ms-grid-column: 2; -ms-grid-row: 1">
    <label for="button">Button:</label>
    <button id="button" value="Button">Clickety Click</button>
  </div>
  <div style="-ms-grid-column: 1; -ms-grid-row: 2">
    <label for="radio">Radio:</label>
    <input type="radio" id="radio" name="radio" />
      <label for="radio">One</label>
    <input type="radio" id="radio" name="radio" />
      <label for="radio">Two</label>
    <input type="radio" id="radio" name="radio" />
      <label for="radio">Three</label>
  </div>
  <div style="-ms-grid-column: 2; -ms-grid-row: 2">
    <label for="select">Select:</label>
    <select id="select">
      <option value="One">One</option>
      <option value="Two">Two</option>
      <option value="Three">Three</option>
     </select>
  </div>
  <div style="-ms-grid-column: 2; -ms-grid-row: 3">
    <label for="rating">Rating:</label>
      <div id="rating" data-win-control="WinJS.UI.Rating"></div>
  </div>
  <div style="-ms-grid-column: 1; -ms-grid-row: 3">
    <label for="toggle">Toggle:</label>
    <div id="toggle" data-win-control="WinJS.UI.ToggleSwitch"></div>
  </div>

  <div style="-ms-grid-column: 1; -ms-grid-row: 4">
    <label for="datepicker">Date Picker:</label>
    <div id="datepicker" data-win-control="WinJS.UI.DatePicker"></div>
  </div>
 <div style="-ms-grid-column: 2; -ms-grid-row: 4">
    <label for="timepicker">Time Picker:</label>
    <div id="timepicker" data-win-control="WinJS.UI.TimePicker"></div>
  </div>
</div>
```

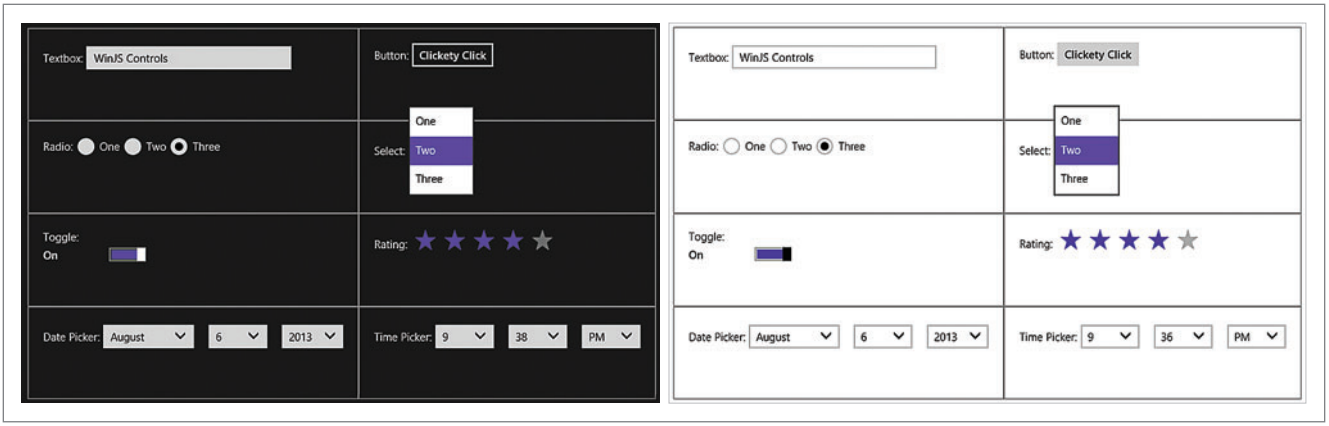


Figure 2 **Common HTML and WinJS Controls in Dark and Light Default Styles**

For a detailed look at the WinJS control landscape, see my column, "Mastering Controls and Settings in Windows Store Apps Built with JavaScript," at msdn.microsoft.com/magazine/dn296546. In the meantime, I'll look at styling guidelines for a few important controls: AppBars, Flyouts and ListViews.

AppBars AppBars are essential UI components of any Windows Store app. CSS selectors exist for styling every aspect of an AppBar, including the AppBar as a whole, and the individual Button labels, images, and tooltips for both regular and hover states. This means you get total control over the look and feel of AppBars. Once you've styled the AppBar itself by overwriting the .win-appbar class, you can move on to the individual commands. Because AppBar Buttons are WinJS controls, the data-win-options attribute contains all the necessary information to properly set up command Buttons by setting the label, icon, section and tooltip properties:

```
<div id="appbar" class="win-appbar"
  data-win-control="WinJS.UI.AppBar">
  <button data-win-control="WinJS.UI.AppBarCommand"
    data-win-options="{
      id:'addFriend',
      label:'Add Friend',
      icon:'addfriend',
      section:'global',
      tooltip:'Add a friend'}"
    type="button">
  </button>
</div>
```

You can make the AppBar look vastly different from the simple default stripe across the bottom of the screen by tweaking the styles of the following class selectors:

- .win-appbar: Styles the AppBar as a whole.
- .win-command: Styles an individual AppBar Button.
- .win-commandimage: Styles the command Button image.
- .win-commandring: Styles the ring around the AppBar Button.
- .win-commandimage:hover and .win-commandring:hover: Style the command Button image and ring hover states.

As you can see, there are styles for every aspect of the AppBar.

Flyouts Flyouts are interactive pop-out windows. In Windows Store apps, you usually implement Flyouts to collect user input, often as part of the app's Settings charm. Flyouts are a great way to implement settings or hold some data the user infrequently accesses. For example, you can use them to provide a privacy statement, required for acceptance of an app into the Windows Store. Styling a Flyout is as easy as overwriting the .win-flyout selector. The system CSS contains settings for .max-width and .max-height, so if you need to collect an entire form of data, you might need to adjust these.

The Flyout is a element that stays hidden and out of sight until the user invokes it from a charm or AppBar command, and then it pops out onto the screen. Though other controls live inside of Flyouts, styling those controls doesn't change, because they're just HTML.

ListView Quite possibly the most complex WinJS control, the ListView offers a way to display multiple data items in a grid or list. ListViews also respond to touch and mouse events initiated by the user, allowing the user to select an item or invoke an action. Starting in Windows 8.1, the ListView control lets users drag, drop and reorder items.

In Windows Store apps, you usually implement Flyouts to collect user input, often as part of the app's Settings charm.

Before working with the ListView control, you must know how it works. The ListView control involves four major moving parts: the ListView itself, and the viewport, surface area and items. The items exist inside the surface area, which is the scrollable region of the ListView. The surface area exists and moves within the viewport, so it makes sense that the viewport has the scrollbars, as illustrated in **Figure 3**.

Together, the viewport and its surface area full of items form the ListView control. Because the ListView control contains these operationally distinct parts, there are many detailed guidelines for styling it, outlined in the Windows Dev Center at bit.ly/HopfUg. For now, I'll look at the most important things to know about styling the ListView control:

- .win-listview: Styles the entire ListView.
- .win-viewport: Styles the ListView's viewport.
- .win-surface: Styles the scrollable area of the ListView.

These moving parts let you do things such as apply a background image that scrolls along with the items. When the surface area is

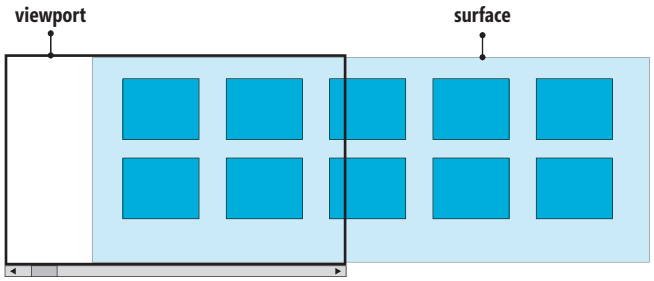


Figure 3 **The Viewport and Its Surface Area Combine to Make the ListView Control**

bigger than the viewport, the viewport shows scrollbars (vertical or horizontal, as needed).

Styling ListView items is straightforward. There are .win-item and .win-container classes to use. Each item in a ListView goes in a container that holds the item's image and text fields. This container is really just an HTML template defining the elements that join to create an item in a ListView. The code in **Figure 4** demonstrates how to define that template and its corresponding ListView. It creates a WinJS.Binding.Template control, which the ListView uses to display data.

As you can see in **Figure 4**, there are elements that make up the ListView's item template built into the Grid and Split project templates. At run time, the app's execution engine injects the .win-container and .win-item classes into these elements, so you won't see those selectors in the code during development, but you can overwrite them in the CSS file to apply your own styles. In the project template's ListView code, .item, .item-image, .item-title, .item-subtitle and .item-overlay are all available as selectors you can modify to specify the look and feel of each individual item inside the grid.

No list or grid is complete without the ability to sort, group and perform other actions that affect its members. So of course there are styles for grouped items as well as individual ones. Style the group headings by overwriting the .win-groupheader class selector and show progress in style by overwriting .win-progress.

Fluid CSS Is for Responsive UIs

The Grid, Split and Navigation projects in Visual Studio contain WinJS CSS selectors that work in tandem with HTML5 semantic elements to ensure a fluid and modern layout. The code in **Figure 5** shows the CSS and HTML required to create the grid layout from the Grid template. Notice the .fragment class defines rows and columns for a grid using the -ms-grid-columns and -ms-grid-rows prefixes. It then applies these prefixes and other selectors to elements in **Figure 5**. The semantic <header> and <section> elements clearly define what type of content goes in each.

Because this CSS is part of the Grid and Split project templates, you don't have to worry about taking on the tedious task of creating a fluid grid that changes based on changes in data or the app's state—it's built-in.

Windows Store JavaScript app templates use the CSS box model (see my blog post, "A guide to element display and positioning with the CSS Box Model," at bit.ly/xxATgL) to lay out the page, and the CSS grid layout (see details at the World Wide Web Consortium [W3C] site at bit.ly/14yzx2h) to create a responsive UI. Notice the grid changes to a vertical list and back to a grid upon changing between snap and full views.

Most of the default project templates in Windows Store apps take advantage of CSS media queries to deliver a sophisticated UI that responds to changes in device and browser size, orientation, and resolution. If you want to learn more about media queries, see my blog post, "Create mobile site layouts with CSS Media Queries," at bit.ly/1c39mDx.

Another reason Windows Store apps make use of media queries is due to Windows 8 supporting four different view states for apps: full, snap, filled and portrait. When users put an app into snap or filled view, it's conceptually the same as accessing the app from a different device, because both the dimensions and orientation of the browser change quite drastically.

Upon inspecting the built-in CSS, you'll find many @media rules for the four app view states as well as for high-contrast and accessible viewing states. Here are media queries that detect snap view and fullscreen-portrait states:

```
@media screen and (-ms-view-state: snapped) {}
@media screen and (-ms-view-state: fullscreen-portrait) {}
```

Usually, the same CSS that works for full view works in filled view (three-fourths of the screen) as well. The preceding queries represent the two most popular media features in the realm of Windows Store apps, but there are many more media features on the W3C site at bit.ly/gnza0F.

> Most of the default project templates in Windows Store apps take advantage of CSS media queries to deliver a sophisticated UI.

Windows Store Apps Use Efficient and Organized CSS

Look at the Visual Studio templates' built-in CSS selectors and you'll notice they're sharp and precise. Specificity in CSS (in any code, really) results in easier-to-maintain and better-performing apps. When you see ".fragment header[role=banner]" in code, you know exactly to which elements it applies—in this case, a <header> element that has a role attribute with a value of "banner" inside an element classed as "fragment." Keep in mind that browsers parse CSS from right to left, so the most specific selectors should go on the right side so the browser can find the matching DOM element faster.

Rather than using script to toggle visual indicators, taking advantage of certain CSS features such as pseudo-selectors for hover, active, disabled, enabled and other visual states ensures easier maintenance. WinJS defines pseudo-selectors as needed for many controls. For example, TextBoxes and CheckBoxes can switch between disabled and enabled states, and anchors can switch between hover and active states. There are even more CSS

animations in Windows 8.1 (bit.ly/KDVSPU), so check them out before turning to JavaScript to do the same animations.

Efficiency means proper file organization as well as efficient code. A file named default.css lives in the \css folder, and all built-in HTML pages contain references to it. The CSS you see in **Figure 5** lives in this file, alongside regular style rules as well as media queries. You can modify or delete default.css, unlike the ui-dark.css and ui-light.css files.

Figure 4 **Code to Define a Template and Its Corresponding ListView**

```
<div id='listviewtemplate' class="itemtemplate"
  data-win-control="WinJS.Binding.Template">
  <div class="item">
    <img class="item-image" src="#"
      data-win-bind="src: backgroundImage; alt: title" />
    <div class="item-overlay">
      <h4 class="item-title" data-win-bind="textContent: title"></h4>
      <h6 class="item-subtitle win-type-ellipsis"
        data-win-bind="textContent: subtitle"></h6>
    </div>
  </div>
</div>
<div class="groupeditemslist win-selectionstylefilled"
  aria-label="List of groups"
  data-win-control="WinJS.UI.ListView"
  data-win-options="{ selectionMode: 'none' }">
</div>
```

Figure 5 **The Foundational HTML and CSS for a Page Fragment**

```
.fragment {
  /* Define a grid with rows for a banner and a body */
  display: -ms-grid;
  -ms-grid-columns: 1fr;
  -ms-grid-rows: 128px 1fr;

  height: 100%;
  width: 100%;
}
  .fragment header[role=banner] {
    /* Define a grid with columns for the back button and page title. */
    display: -ms-grid;-ms-grid-columns: 39px 81px 1fr;
    -ms-grid-rows: 1fr;

  }
  .fragment header[role=banner] .win-backbutton {
    -ms-grid-column: 2;
    margin-top: 59px;
  }
  .fragment header[role=banner] .titlearea {
    -ms-grid-column: 3;
    margin-top: 37px;
  }
  .fragment header[role=banner] .titlearea .pagetitle {
    width: calc(100% - 20px);
  }
  .fragment section[role=main] {
    -ms-grid-row: 2;
    height: 100%;
    width: 100%;
  }
<div class="fragment groupeditemspage">
  <header aria-label="Header content" role="banner">
    <button class="win-backbutton" aria-label="Back"
      disabled type="button"></button>
    <h1 class="titlearea win-type-ellipsis">
      <span class="pagetitle">CSS in Windows Store apps</span>
    </h1>
  </header>
  <section aria-label="Main content" role="main">
    <div class="groupeditemslist win-selectionstylefilled"
      aria-label="List of groups"
    data-win-control="WinJS.UI.ListView"
      data-win-options="{ selectionMode: 'none' }">
  </div>
  </section>
</div>
```

Visual Studio project templates organize files into folders that contain related items, such as groupedItems.html, groupedItems.js and groupedItems.css in the \pages\groupedItems folder alongside other similar structures. You can continue to organize files into folders this way, but you can also rearrange the files in any way you'd like. Perhaps you're reorganizing the project structure because you want to try using the Model-View-ViewModel (MVVM) pattern, and in that case, CSS files could go into a different folder. Maybe you want to toss all the CSS into one folder instead. It doesn't really matter, so long as your file organization makes sense for your project and it's easy to maintain.

For readability's sake, break out CSS into separate files arranged by their functionality or features. For example, you might have CSS for a single control used by many pages, such as a Flyout. That file can go in a folder for common CSS or it can go in the same folder as the Flyout. There's nothing worse than plowing through a CSS file filled with CSS that has nothing to do with the pages that reference it. For example, CSS that meets the following criteria deserves its own file:

- CSS for a specific Flyout control or settings page.
- CSS containing just colors and aesthetics, as this allows you to create themes. You can then swap in and out CSS files like color swatches on an as-needed basis.
- CSS in media queries for a specific scenario such as snap or filled views. This CSS can go in its own file in the same directory as the page that references it.

This way you can use each functional unit of CSS as you need rather than having to load all the CSS for every page whether the page uses the CSS or not. Remember, these are only guidelines and every project is unique and may have different needs.

Windows Store app projects execute code that's local to the device rather than downloading the CSS or script first, so there's no need to bundle and minify CSS and scripts like you would in a Web project. Of course, the app must load the code into memory before executing, so you still want to avoid bloated code for performance and readability reasons.

Focus on Building the App

Styling the many built-in WinJS controls is a snap because the framework provides cutting-edge CSS3 and HTML5 features. The core CSS in WinJS creates a modern, fluid and flexible UI that responds to changes in app view state when the user switches between full and snap views. Having the core WinJS CSS available out of the box means you can focus on building the app rather than fiddling with structural CSS to create a modern-style UI. ■

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 expert for reviewing this article: *Eric Schmidt (Microsoft)*

CHARLES PETZOLD

Text Formatting and Scrolling with DirectWrite

The display of text in computer graphics has always been rather awkward. It would be great to display text with as much ease as other two-dimensional graphics, such as geometries and bitmaps, yet text comes with hundreds of years of baggage and some very definite needs—the crucial requirement of readability, for example.

In recognition of the special nature of text within graphics, DirectX splits the job of working with text into two major subsystems, Direct2D and DirectWrite. The ID2D1RenderTarget interface declares the methods for displaying text, along with other 2D graphics, while interfaces beginning with IDWrite help prepare the text for display.

At the border between graphics and text are some interesting techniques, such as obtaining outlines of text characters, or implementing the IDWriteTextRenderer interface for intercepting and manipulating text on its way to the display. But a necessary preliminary is a solid understanding of basic text formatting—in other words, the display of text that's intended to be read rather than admired with aesthetic rapture.

The Simplest Text Output

The most fundamental text display interface is IDWriteTextFormat, which combines a font family (Times New Roman or Arial, for example) along with a style (italic or oblique), weight (bold or light), stretch (narrow or expanded), and a font size. You'll probably define a reference-counted pointer for the IDWriteTextFormat object as a private member in a header file:

```
Microsoft::WRL::ComPtr<IDWriteTextFormat> m_textFormat;
```

The object is created with a method defined by IDWriteFactory:

```
dwriteFactory->CreateTextFormat(
  L"Century Schoolbook", nullptr,
  DWRITE_FONT_WEIGHT_NORMAL,
  DWRITE_FONT_STYLE_ITALIC,
  DWRITE_FONT_STRETCH_NORMAL,
  24.0f, L"en-US", &m_textFormat);
```

In the general case, your application will probably create several IDWriteTextFormat objects for different font families, sizes and styles. These are device-independent resources, so you can create them at any time after you call DWriteCreateFactory to obtain an IDWriteFactory object, and keep them for the duration of your application.

If you spell the family name incorrectly—or if a font with that name isn't on your system—you'll get a default font. The second argument indicates the font collection in which to search for such a font. Specifying nullptr indicates the system font collection. You can also have private font collections.

The size is in device-independent units based on a resolution of 96 units per inch, so a size of 24 is equivalent to an 18-point font. The language indicator refers to the language of the font family name, and can be left as an empty string.

Once you've created an IDWriteTextFormat object, all the information you've specified is immutable. If you need to change the font family, style or size, you'll need to re-create the object.

What more do you need to render text beyond the IDWriteTextFormat object? Obviously, the text itself, but also the location on the screen where you want it to be displayed, and the color. These items are specified when the text is rendered: The destination of the text is indicated not with a point but with a rectangle of type D2D1_RECT_F. The color of the text is specified with a brush, which can be any type of brush, such as a gradient brush or image brush.

Here's a typical DrawText call:

```
deviceContext->DrawText(
  L"This is text to be displayed",
  28,    // Characters
  m_textFormat.Get(),
  layoutRect,
  m_blackBrush.Get(),
  D2D1_DRAW_TEXT_OPTIONS_NONE,
  DWRITE_MEASURING_MODE_NATURAL);
```

By default, the text is broken into lines and wrapped based on the width of the rectangle (or the height of the rectangle for languages that read from top to bottom). If the text is too long to display within the rectangle, it will continue beyond the bottom. The penultimate argument can indicate optional flags to clip text falling outside the rectangle, or to not align characters on pixel boundaries (which is useful if you'll be performing animations on the text) or, in Windows 8.1, to enable multicolor font characters.

The downloadable code for this column includes a Windows 8.1 program that uses IDWriteTextFormat and DrawText to display Chapter 7 of Lewis Carroll's "Alice's Adventures in Wonderland." (I obtained the text from the Project Gutenberg Web site, but modified it somewhat to make it more consistent with the typography of the original edition.) The program is called PlainTextAlice, and I created it using the DirectX App (XAML) template in Visual Studio Express 2013 Preview for Windows 8.1. This project template generates a XAML file containing a SwapChainPanel and all the necessary overhead for displaying DirectX graphics on it.

The file with the text is part of the project content. Each paragraph is a single line, and each is separated by a blank line. The DirectXPage

Code download available at archive.msdn.microsoft.com/mag201310DXF.

Figure 1 **The PlainTextAliceRenderer.cpp File**

```
#include "pch.h"
#include "PlainTextAliceRenderer.h"

using namespace PlainTextAlice;

using namespace D2D1;
using namespace Platform;

PlainTextAliceRenderer::PlainTextAliceRenderer(
  const std::shared_ptr<DeviceResources>& deviceResources) :
  m_text(L""),
  m_deviceResources(deviceResources)
{
  m_deviceResources->GetDWriteFactory()->
    CreateTextFormat(L"Century Schoolbook",
                     nullptr,
                     DWRITE_FONT_WEIGHT_NORMAL,
                     DWRITE_FONT_STYLE_NORMAL,
                     DWRITE_FONT_STRETCH_NORMAL,
                     24.0f,
                     L"en-US",
                     &m_textFormat);

  CreateDeviceDependentResources();
}

void PlainTextAliceRenderer::CreateDeviceDependentResources()
{
  m_deviceResources->GetD2DDeviceContext()->
    CreateSolidColorBrush(D2D1::ColorF(D2D1::ColorF::Black),
                          &m_blackBrush);

  m_deviceResources->GetD2DFactory()->
    CreateDrawingStateBlock(&m_stateBlock);
}

void PlainTextAliceRenderer::CreateWindowSizeDependentResources()
{
  Windows::Foundation::Size windowBounds =
    m_deviceResources->GetOutputBounds();
  m_layoutRect = RectF(50, 0, windowBounds.Width - 50,  windowBounds.Height);
}

void PlainTextAliceRenderer::ReleaseDeviceDependentResources()
{
  m_blackBrush.Reset();
  m_stateBlock.Reset();
}

void PlainTextAliceRenderer::SetAliceText(std::wstring text)
{
  m_text = text;
}

void PlainTextAliceRenderer::Render()
{
  ID2D1DeviceContext* context = m_deviceResources->GetD2DDeviceContext();

  context->SaveDrawingState(m_stateBlock.Get());
  context->BeginDraw();
  context->Clear(ColorF(ColorF::White));
  context->SetTransform(m_deviceResources->GetOrientationTransform2D());
  context->DrawText(m_text.c_str(),
                    m_text.length(),
                    m_textFormat.Get(),
                    m_layoutRect,
                    m_blackBrush.Get(),
                    D2D1_DRAW_TEXT_OPTIONS_NONE,
                    DWRITE_MEASURING_MODE_NATURAL);

  HRESULT hr = context->EndDraw();
  context->RestoreDrawingState(m_stateBlock.Get());
}
```

class loads the text in a Loaded event handler and transfers it to the PlainTextAliceMain class (created as part of the project), which transfers it to the PlainTextAliceRenderer class—the class I contributed to the project.

Because the graphics displayed by this program are fairly static, I disabled the rendering loop in DirectXPage by not attaching a handler for the CompositionTarget::Rendering event. Instead, PlainTextAliceMain determines when the graphics should be redrawn, which is only when the text is loaded or when the application window changes size or orientation. At these times, PlainTextAliceMain calls the Render method in PlainTextAliceRenderer and the Present method in DeviceResources.

The C++ part of the PlainTextAliceRenderer class is shown in **Figure 1**. For clarity, I've removed the HRESULT checks.

Notice that the m_layoutRect member is calculated based on the application's size on the screen but with a 50-pixel margin at the left and right. The result is shown in **Figure 2**.

First off, I'll find some good things to say about this program: Clearly, with a minimum overhead, the text is properly wrapped into nicely separated paragraphs.

The inadequacies of the PlainTextAlice project are also obvious: The spacing between the paragraphs exists only because the original text file has blank lines I inserted just for that purpose. If you wanted a somewhat smaller or wider paragraph spacing, that wouldn't be possible. Moreover, there's no way to indicate italicized or boldfaced words within the text.

But the biggest drawback is that you can see only the beginning of the chapter. Scrolling logic could be implemented, but how can you tell how far to scroll? The really big problem with IDWriteTextFormat and DrawText is that the height of the formatted rendered text is simply not available.

In conclusion, using DrawText makes sense only when the text has uniform formatting, and you know the rectangle you're specifying is sufficient to fit the text or you don't care if there's a little runover. For more sophisticated purposes a better approach is required.

CHAPTER VII.

A MAD TEA-PARTY

There was a table set out under a tree in front of the house, and the March Hare and the Hatter were having tea at it: a Dormouse was sitting between them, fast asleep, and the other two were using it as a cushion, resting their elbows on it, and talking over its head. "Very uncomfortable for the Dormouse," thought Alice; "only, as it's asleep, I suppose it doesn't mind."

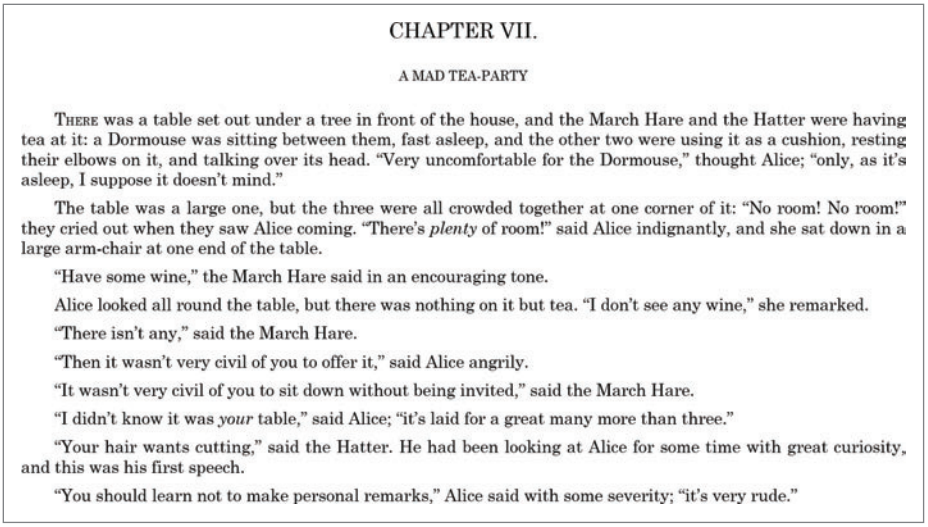
The table was a large one, but the three were all crowded together at one corner of it: "No room! No room!" they cried out when they saw Alice coming. "There's PLENTY of room!" said Alice indignantly, and she sat down in a large arm-chair at one end of the table.

"Have some wine," the March Hare said in an encouraging tone.

Alice looked all round the table, but there was nothing on it but tea. "I don't see any wine," she remarked.

"There isn't any," said the March Hare.

Figure 2 **The PlainTextAlice Program**



CHAPTER VII.

A MAD TEA-PARTY

There was a table set out under a tree in front of the house, and the March Hare and the Hatter were having tea at it: a Dormouse was sitting between them, fast asleep, and the other two were using it as a cushion, resting their elbows on it, and talking over its head. "Very uncomfortable for the Dormouse," thought Alice; "only, as it's asleep, I suppose it doesn't mind."

The table was a large one, but the three were all crowded together at one corner of it: "No room! No room!" they cried out when they saw Alice coming. "There's *plenty* of room!" said Alice indignantly, and she sat down in a large arm-chair at one end of the table.

"Have some wine," the March Hare said in an encouraging tone.

Alice looked all round the table, but there was nothing on it but tea. "I don't see any wine," she remarked.

"There isn't any," said the March Hare.

"Then it wasn't very civil of you to offer it," said Alice angrily.

"It wasn't very civil of you to sit down without being invited," said the March Hare.

"I didn't know it was *your* table," said Alice; "it's laid for a great many more than three."

"Your hair wants cutting," said the Hatter. He had been looking at Alice for some time with great curiosity, and this was his first speech.

"You should learn not to make personal remarks," Alice said with some severity; "it's very rude."

Figure 3 **The ParagraphFormattedAlice Program**

Before moving on, take note that the information specified in CreateTextFormat method is immutable in the IDWriteTextFormat object, but the interface declares several methods that let you change how the text is displayed: For example, SetParagraphAlignment alters the vertical placement of the text within the rectangle specified in DrawText, while SetTextAlignment lets you specify whether the lines of the paragraph are left, right, centered or justified within the rectangle. The arguments to these methods use words such as near, far, leading, and trailing to be generalized for text that reads from top to bottom or from right to left. You can also control line spacing, text wrapping and tab stops.

The Text Layout Approach

The next step up from IDWriteTextFormat is a big one, and it's ideal for pretty much all standard text output needs, including hit-testing. It involves an object of type IDWriteTextLayout, which not only derives from IDWriteTextFormat, but incorporates an IDWriteTextFormat object when it's being instantiated.

Here's a reference-counted pointer to an IDWriteTextLayout object, probably declared in a header file:

```
Microsoft::WRL::ComPtr<IDWriteTextLayout> m_textLayout;
```

Create the object like so:

```
dwriteFactory->CreateTextLayout(
  pText, wcslen(pText),
  m_textFormat.Get(),
  maxWidth, maxHeight,
  &m_textLayout);
```

Unlike the IDWriteTextFormat interface, IDWriteTextLayout incorporates the text itself and the desired height and width of the rectangle to format the text. The DrawTextLayout method that displays an IDWriteTextLayout object needs only a 2D point to indicate where the top-left corner of the formatted text is to appear:

```
deviceContext->DrawTextLayout(
  point,
  m_textLayout.Get(),
  m_blackBrush.Get(),
  D2D1_DRAW_TEXT_OPTIONS_NONE);
```

Because the IDWriteTextLayout object has all the information it needs to calculate line breaks before the text is rendered, it also knows how large the rendered text will be. IDWriteTextLayout has several methods—GetMetrics, GetOverhangMetrics, GetLineMetrics and GetClusterMetrics—that provide a wealth of information to help you work with this text effectively. For example, GetMetrics provides the total width and height of the formatted text, as well as the number of lines and other information.

Although the CreateTextLayout method includes maximum width and height arguments, these can be set to other values at a later time. (The text itself is immutable, however.) If the display area changes (for example, you turn your tablet from landscape to portrait mode), you don't need to re-create the IDWriteTextLayout object. Just call the SetMaxWidth and SetMaxHeight methods declared by the interface. Indeed, when you first create the IDWriteTextLayout object, you can set the maximum width and height arguments to zero.

The ParagraphFormattedAlice project uses IDWriteTextLayout and DrawTextLayout, and the results are shown in **Figure 3**. You still can't scroll to see the rest of the text, but notice that some lines are centered, and many have first line indents. The titles use a larger font size than the rest and some words are italicized.

The text file is a little different in this project than the first project: Each paragraph is still a separate line, but no blank lines separate these paragraphs. Rather than using a single IDWriteTextFormat object for the entire text, each paragraph in ParagraphFormattedAlice is a separate IDWriteTextLayout object rendered with a separate DrawTextLayout call. The space between the paragraphs can therefore be set to any desired amount.

To work with the text, I defined a structure named Paragraph:

```
struct Paragraph
{
  std::wstring Text;
  ComPtr<IDWriteTextLayout> TextLayout;
  float TextHeight;
  float SpaceAfter;
};
```

Figure 4 **The SetWidth Method in AliceParagraphGenerator**

```
float AliceParagraphGenerator::SetWidth(float width)
{
  if (width <= 0)
    return 0;

  float totalHeight = 0;

  for (Paragraph& paragraph : m_paragraphs)
  {
    HRESULT hr = paragraph.TextLayout->SetMaxWidth(width);
    hr = paragraph.TextLayout->SetMaxHeight(FloatMax());

    DWRITE_TEXT_METRICS textMetrics;
    hr = paragraph.TextLayout->GetMetrics(&textMetrics);

    paragraph.TextHeight = textMetrics.height;
    totalHeight += paragraph.TextHeight + paragraph.SpaceAfter;
  }
  return totalHeight;
}
```

A helper class called AliceParagraphGenerator generates a collection of Paragraph objects based on the lines of text.

IDWriteTextLayout has a bunch of methods to set formatting on individual words or other blocks of text. For example, here's how the five characters beginning at offset 23 in the text are italicized:

```
DWRITE_TEXT_RANGE range = { 23, 5 };
textLayout->SetFontStyle(DWRITE_FONT_STYLE_ITALIC, range);
```

Similar methods are available for the font family, collection, size, weight, stretch, underline and strikethrough. In real-life applications, text formatting is usually defined by markup (such as HTML), but to keep things simple, the AliceParagraphGenerator class works with a plain-text file and contains hardcoded locations for the italicized words.

AliceParagraphGenerator also has a SetWidth method to set a new display width, as shown in **Figure 4**. (For clarity, the HRESULT checks have been removed from the code in this figure.) The display width changes when the window size changes, or when a tablet changes orientation. SetWidth loops through all the Paragraph objects, calls SetMaxWidth on the TextLayout, and then obtains a new formatted height of the paragraph that it saves in TextHeight. Earlier, the SpaceAfter field was simply set to 12 pixels for most of the paragraphs, 36 pixels for the headings, and 0 for a couple lines of verse. This makes it easy to obtain the height of each paragraph and the total height of all the text in the chapter.

The Render method in ParagraphFormattedAliceRenderer also loops through all the Paragraph objects and calls DrawTextLayout with a different origin based on the accumulated height of the text.

The First-Line Indent Problem

As you can see in **Figure 3**, the ParagraphFormattedAlice program indents the first line of the paragraphs. Perhaps the easiest way to make such an indent is by inserting some blank spaces at the beginning of the text string. The Unicode standard defines codes for an em space (which has a width equal to the point size), en space (half that), quarter em, and smaller spaces, so you can combine these for the desired spacing. The advantage is that the indent is proportional to the point size of the font.

However, that approach won't work for a negative first-line indent—sometimes called a hanging indent—which is a first line that begins to the left of the rest of the paragraph. Moreover, first-line indents are commonly specified in metrics (such as a half-inch) that are independent of the point size.

Figure 5 **Implementing Scrolling in ScrollableAlice**

```
std::vector<Paragraph> paragraphs =
  m_aliceParagraphGenerator.GetParagraphs();
float outputHeight = m_deviceResources->GetOutputBounds().Height;
D2D1_POINT_2F origin = Point2F(50, -m_scrollOffset);

for (Paragraph paragraph : paragraphs)
{
  if (origin.y + paragraph.TextHeight + paragraph.SpaceAfter > 0)
    context->DrawTextLayout(origin, paragraph.TextLayout.Get(),
    m_blackBrush.Get());

  origin.y += paragraph.TextHeight + paragraph.SpaceAfter;

  if (origin.y > outputHeight)
    break;
}
```

For these reasons, I decided instead to implement first-line indents using the SetInlineObject method of the IDWriteTextLayout interface. This method is intended to allow you to put any graphical object inline with the text so it becomes almost like a separate word whose size is taken into account for wrapping the line.

The SetInlineObject method is commonly used for inserting small bitmaps into the text. To use it for that or any other purpose, you need to write a class that implements the IDWriteInlineObject interface, which, besides the three standard IUnknown methods, declares the GetMetrics, GetOverhangMetrics, GetBreakConditions, and Draw methods. Basically, the class you supply is called while the text is being measured or rendered. For my FirstLineIndent class, I defined a constructor with an argument indicating the desired indent in pixels, and that value is basically returned by the GetMetrics call to indicate the size of the embedded object. The class's implementation of Draw does nothing. Negative values work fine for hanging indents.

> I created the ScrollableAlice project in Visual Studio 2013 Preview using the same DirectX App (XAML) template I used for the earlier projects.

You call the SetInlineObject of IDWriteFontLayout just like SetFontStyle and other methods based on text ranges, but it was only when I began using SetInlineObject that I discovered the range couldn't have a length of zero. In other words, you can't simply insert an inline object. The inline object has to replace at least one character of text. For that reason, while defining the Paragraph objects, the code inserts a no-width space character ('\x200B') at the beginning of each line, which has no visual appearance but can be replaced when SetInlineObject is called.

DIY Scrolling

The ParagraphFormattedAlice program doesn't scroll, but it has all the essential information to implement scrolling, specifically, the total height of the rendered text. The ScrollableAlice project demonstrates one approach to scrolling: The program still outputs to a SwapChainPanel the size of the program's window, but it offsets the rendering based on the user's mouse or touch input. I think of this approach as "do it yourself" scrolling.

I created the ScrollableAlice project in Visual Studio 2013 Preview using the same DirectX App (XAML) template I used for the earlier projects, but I was able to take advantage of another interesting aspect of this template. The template contains code in DirectXPage.cpp that creates a secondary thread of execution to handle Pointer events from the SwapChainPanel. This technique avoids bogging down the UI thread with this input.

Of course, introducing a secondary thread complicates things as well. I wanted some inertia in my scrolling, which meant I wanted

Manipulation events rather than Pointer events. This required that I use DirectXPage to create a GestureRecognizer object (also in that secondary thread), which generates Manipulation events from Pointer events.

The previous ParagraphFormattedAlice program redraws the program's window when the text is loaded and when the window changes size. ScrollableAlice does that also, and that redrawing continues to occur in the UI thread. ScrollableAlice also redraws the window when ManipulationUpdated events are fired, but this occurs in the secondary thread created for the pointer input.

What happens if you give the text a good flick with your finger so it continues scrolling with inertia, and while the text is still scrolling, you resize the window? There's a good possibility that overlapping DirectX calls will be made from two different threads at the same time, and that's a problem.

What's required is some thread synchronization, and a good, easy solution involves the critical_section class in the Concurrency namespace. In a header file, the following is declared:

```
Concurrency::critical_section m_criticalSection;
```

The critical_section class contains an embedded class named scoped_lock. The following statement creates an object named lock of type scoped_lock by calling the constructor with the critical_section object:

```
critical_section::scoped_lock lock(m_criticalSection);
```

This constructor assumes ownership of the m_criticalSection object, or blocks execution if the m_criticalSection object is owned by another thread. What's nice about this scoped_lock class is that the destructor releases ownership of the m_criticalSection when the lock object goes out of scope, so it's incredibly easy to just sprinkle a bunch of these around in various methods that might be called concurrently.

I determined it would be easiest to implement this critical section in DirectXPage, which contains some crucial calls to the DeviceResources class (such as UpdateForWindowSizeChanged) that require other threads be blocked for the duration. Although it's not a good idea to block the UI thread (which happens when pointer events are fired), these blocks are extremely short.

By the time the scrolling information is delivered to the ScrollableAliceRenderer class, it's in the form of a floating-point value stored as a variable named m_scrollOffset, clamped between 0 and a maximum value equal to the difference between the height of the full chapter of text and the height of the window. The Render method uses that value to determine how to begin and end the display of paragraphs, as shown in **Figure 5**.

Scrolling with a Bounce

Although the ScrollableAlice program implements scrolling that has touch inertia, it doesn't have the characteristic Windows 8 bounce when you try to scroll past the top or bottom. That (by now) familiar bounce is incorporated into ScrollViewer, and while it might be fun to try to duplicate the ScrollViewer bounce in your own code, that's not the job for today.

Because scrollable text might be long, it's best not to try to render it all on one surface or bitmap. DirectX has restrictions on how large these surfaces can be. The ScrollableAlice program gets around these restrictions by limiting its display to a SwapChainPanel the size of the program's window, and that works just fine. But for ScrollViewer to work, the content must have a size in layout reflecting the full height of the formatted text.

Fortunately, Windows 8 supports an element that does exactly what's needed. The VirtualSurfaceImageSource class derives from SurfaceImageSource, which in turn derives from ImageSource so it can serve as a bitmap source for an Image element in a ScrollViewer. VirtualSurfaceImageSource can be any desired size (and can be resized without being re-created), and gets around DirectX size limitations by virtualizing the surface area and implementing on-demand drawing. (However, SurfaceImageSource and VirtualSurfaceImageSource are not optimal for high-performance frame-based animations.)

> Because scrollable text might be long, it's best not to try to render it all on one surface or bitmap.

VirtualSurfaceImageSource is a Windows Runtime ref class. To use it in conjunction with DirectX, it must be cast to an object of type IVirtualSurfaceImageSourceNative, which reveals the methods used to implement on-demand drawing. These methods report what rectangular areas need to be updated, and allow the program to be notified of new update rectangles by supplying a class that implements IVirtualSurfaceUpdatesCallbackNative.

The project demonstrating this technique is BounceScrollableAlice, and because it didn't require a SwapChainPanel, I created it in Visual Studio 2013 Preview based on the Blank App (XAML) template. For the required class that implements IVirtualSurfaceUpdatesCallbackNative I created a class named VirtualSurfaceImageSourceRenderer, and it also provides much of the DirectX overhead. The AliceVsisRenderer class derives from VirtualSurfaceImageSourceRenderer to provide the Alice-specific drawing.

The update rectangles available from IVirtualSurfaceImageSourceNative are relative to the full size of the VirtualSurfaceImageSource, but drawing coordinates are relative to the update rectangles. This means the DrawTextLayout calls in BounceScrollableAlice are virtually the same as those shown in **Figure 5**, except the initial origin is set to the negative of the top of the update rectangle rather than the scroll offset, and the outputHeight value is the difference between the bottom and top of the update rectangle.

By introducing ScrollViewer into the mix, the text display feels truly Windows 8-like, and you can even use a pinch gesture to make it larger or smaller, emphasizing even more that this Windows 8 melding of DirectX and XAML provides something close to the best of both worlds. ■

Charles Petzold *is a longtime contributor to* MSDN Magazine *and the author of "Programming Windows, 6th edition" (Microsoft Press, 2012), a book about writing applications for Windows 8. His Web site is charlespetzold.com.*

Thanks to the following technical experts for reviewing this article: *Jim Galasyn (Microsoft) and Justin Panian (Microsoft)*

LIVE EVENT

SharePoint LIVE!

TRAINING FOR COLLABORATION

ORLANDO
NOVEMBER 18-22, 2013
LOEWS ROYAL PACIFIC RESORT
AT UNIVERSAL

COLLABORATE AND LISTEN

Whether you've implemented SharePoint 2013, or you're still using and supporting older versions, SharePoint Live! is THE place to gather and learn how to customize, deploy and maintain SharePoint Server and SharePoint foundation to maximize business value.

REGISTER TODAY TO SAVE $300!

Super Early Bird Savings End October 9.
Use Promo Code SPOCT1

LIVE! 360

IT EVENTS WITH PERSPECTIVE

Four co-located events; one low price!

- SharePoint Live!
- Visual Studio Live! Orlando
- SQL Server Live!
- Modern Apps Live!

live360events.com

Scan the QR code for more information on SharePoint Live!

SUPPORTED BY
Microsoft SharePoint Visual Studio msdn magazine Visual Studio MAGAZINE

PRODUCED BY
1105 MEDIA

Remaking Higher Education

I was saving this topic for its customary January slot, but events overtook me. About 18 months ago, I reported on the first computer science massive open online course (MOOC), Sebastian Thrun's AI class at Stanford (msdn.microsoft.com/magazine/hh708761). Then six months ago, I reported that MOOCs were becoming popular, but had yet to replace regular college courses because they failed to award credit toward a degree (msdn.microsoft.com/magazine/jj883963). I predicted this would change quickly because of the compelling economics. It has, and I can't wait to tell you about it.

Georgia Tech is now offering an online master's degree in computer science, using MOOCs (see b.gatech.edu/140FDnH and slate.me/163CyTJ). The tuition is a thrifty $6,600, compared to $45,000 for the school's conventional program. You won't get much face time with the faculty, but for that price, who cares? How much face time did you ever get in college, and how much was it worth? (For me, outside the physics department, damn little and damn little.)

Jonathan Rees, professor of history at Colorado State, thinks MOOCs are evil (slate.me/1499qaL). I think he's more worried about the loss of his tenured job and perks, and about having to deal with economic forces from which he has hitherto sheltered in academia. Welcome to the real world like the rest of us, pal.

Worse, I think he's narrow-minded and not looking at how the greater availability of good, cheap education would benefit the world—not just in lowering the cost and improving the quality of conventional college, but in ways no one has yet thought. Imagine a MOOC on poetry stimulating the minds of people in an old folks' home. Or prison inmates learning a skill other than license-plate making, so they have some chance of employment when they get out. Or enrichment for smart high school students like my daughters will be. Or … you get the idea.

Rees reminds me of the movie executives in the early 1980s who tried, unsuccessfully, to block VCRs. For another of my beloved analogies, compare movies to live theater. The latter costs less to produce initially, but each performance requires expensive actors and musicians in an expensive, dedicated building. The former costs more to make, but each performance costs almost nothing, so you can spread that cost over as many units as you can sell through as many channels as you can stuff them. A movie ticket in New York City costs $10, a musical theater ticket costs $100. They sell a lot more movie tickets. The same economics will apply to MOOCs versus live-delivery education. Georgia Tech expects to attract enough MOOC students to make a profit in its first year.

I've always favored creative destruction. I've never mourned for the typewriter manufacturers or travel agents or encyclopedia salesmen that I've helped put out of business. But then I've always been the creative destructor, never the creatively destructed.

I'm pondering how to adjust my own offerings. What makes my online Harvard Extension class on .NET worth $2,000? MOOCs don't offer direct contact with the instructor, because that can't scale. So I'm inviting every student to Skype me personally at the start of the class—that way I know who they are and what they're doing, where they're coming from, and to where they're trying to get. I used to do that before distance learning, when all the students came to the auditorium and I could talk to them in person. I flatter myself that face time with me, as opposed to some anonymous droning grad student, is worth something. I wonder how a contact buzz would propagate through an online group beer session.

I've always favored creative destruction. I've never mourned for the typewriter manufacturers or travel agents or encyclopedia salesmen that I've helped put out of business.

Will Georgia Tech be able to maintain quality in its online program? I think so. The school is partnering with Thrun's company Udacity, which knows more about running MOOCs than anyone else. There are still some challenges to figure out, as always at this stage. Detecting cheating is a big one, as is connecting graduates to potential employers. But as I said in my first article, with these compelling economics, the challenges will get solved.

If I were a traditional four-year residential college, I'd be quaking right now. But as the father of a teenager (see msdn.microsoft.com/magazine/dn385715), soon to be a college student, I'm really happy to see this first falling boulder of an extremely large avalanche. ■

DAVID S. PLATT *teaches programming .NET at Harvard University Extension School and at companies all over the world. He's the author of 11 programming books, including "Why Software Sucks" (Addison-Wesley Professional, 2006) and "Introducing Microsoft .NET" (Microsoft Press, 2002). Microsoft named him a Software Legend in 2002. He wonders whether he should tape down two of his daughter's fingers so she learns how to count in octal. You can contact him at rollthunder.com.*

.NET TOOLS FOR 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.

DOWNLOAD YOUR FREE TRIAL

componentone.com/se

ComponentOne
a division of GrapeCity