THE MICROSOFT JOURNAL FOR DEVELOPERS

FEBRUARY 2015 VOL 30 NO 2

# msdn magazine

THE MICROSOFT JOURNAL FOR DEVELOPERS

FEBRUARY 2015 VOL 30 NO 2

# msdn magazine





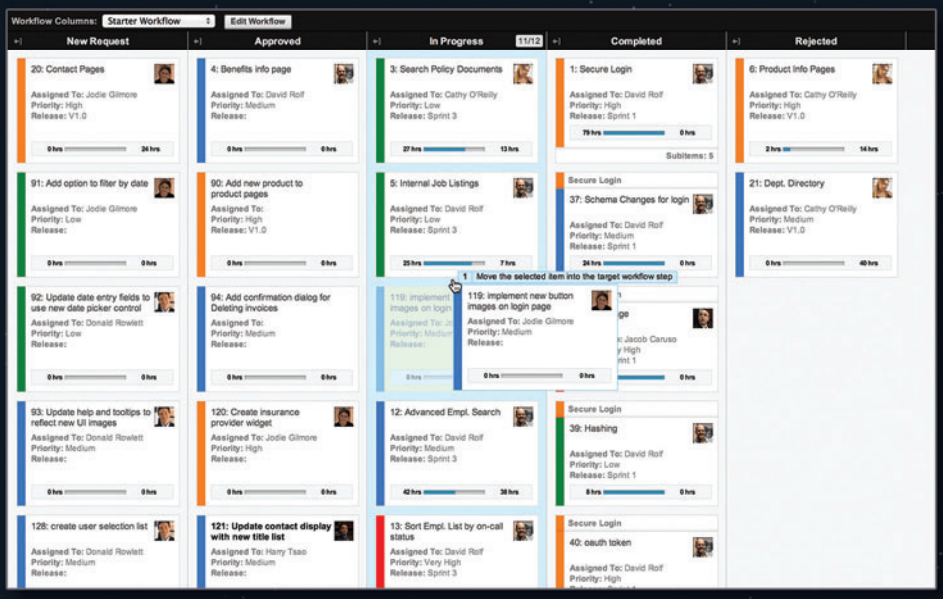
## COLUMNS



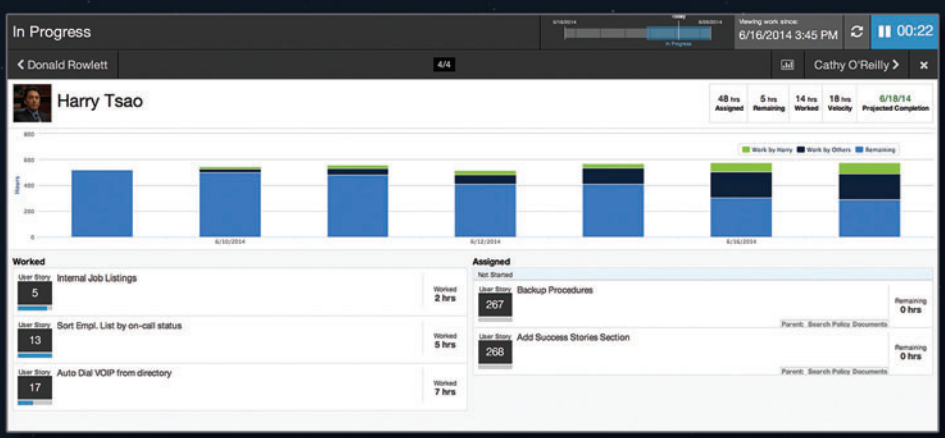
Microsoft

# Don’t be left in the dark when it comes to your team’s development progress.

Axosoft empowers you and your team with instant visibility.




Check out item status and progress at a glance with our Kanban board view




View team members’ work capacity and assignments in our Daily Scrum mode

- Track user stories & defects
- Create custom process workflows
- Log work spent on each item
- Plan out releases and sprints
- Manage users, teams, & project roles
- Flexible notification system
- Help desk and self-service portal
- Powerful reporting tools

Integrates with popular tools:

salesforce slack Visual Studio zendesk and more


Automated burndown charts, velocity, projected ship dates and more

Let us illuminate how we'll help you ship on time and on budget!

Try us free today at **axosoft.com/MSDN**

axosoft The #1 Scrum Software
800.653.0024

msdn magazine

FEBRUARY 2015 VOLUME 30 NUMBER 2



**ID STATEMENT** 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, 9201 Oakdale Ave. Ste 101, Chatsworth, CA 91311 www.1105media.com

**MEDIA KITS** Direct your Media Kit requests to Chief Revenue Officer Dan LaBianca, 972-687-6702 (phone), 972-687-6799 (fax), dlabianca@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

**Reaching the Staff**

Staff may be reached via e-mail, telephone, fax, or mail. A list of editors and contact information is also available online at Redmondmag.com.

E-mail: To e-mail any member of the staff, please use the following form: FirstinitialLastname@1105media.com

Irvine Office (weekdays, 9:00 a.m. – 5:00 p.m. PT)
Telephone 949-265-1520; Fax 949-265-1528
4 Venture, Suite 150, Irvine, CA 92618

Corporate Office (weekdays, 8:30 a.m. – 5:30 p.m. PT)
Telephone 818-814-5200; Fax 818-734-1522
9201 Oakdale Avenue, Suite 101, Chatsworth, CA 91311

The opinions expressed within the articles and other contentsherein do not necessarily express those of the publisher.

Microsoft

BPA WORLDWIDE

1906 100 2006 AMERICAN BUSINESS MEDIA

WPA Western Publications Association

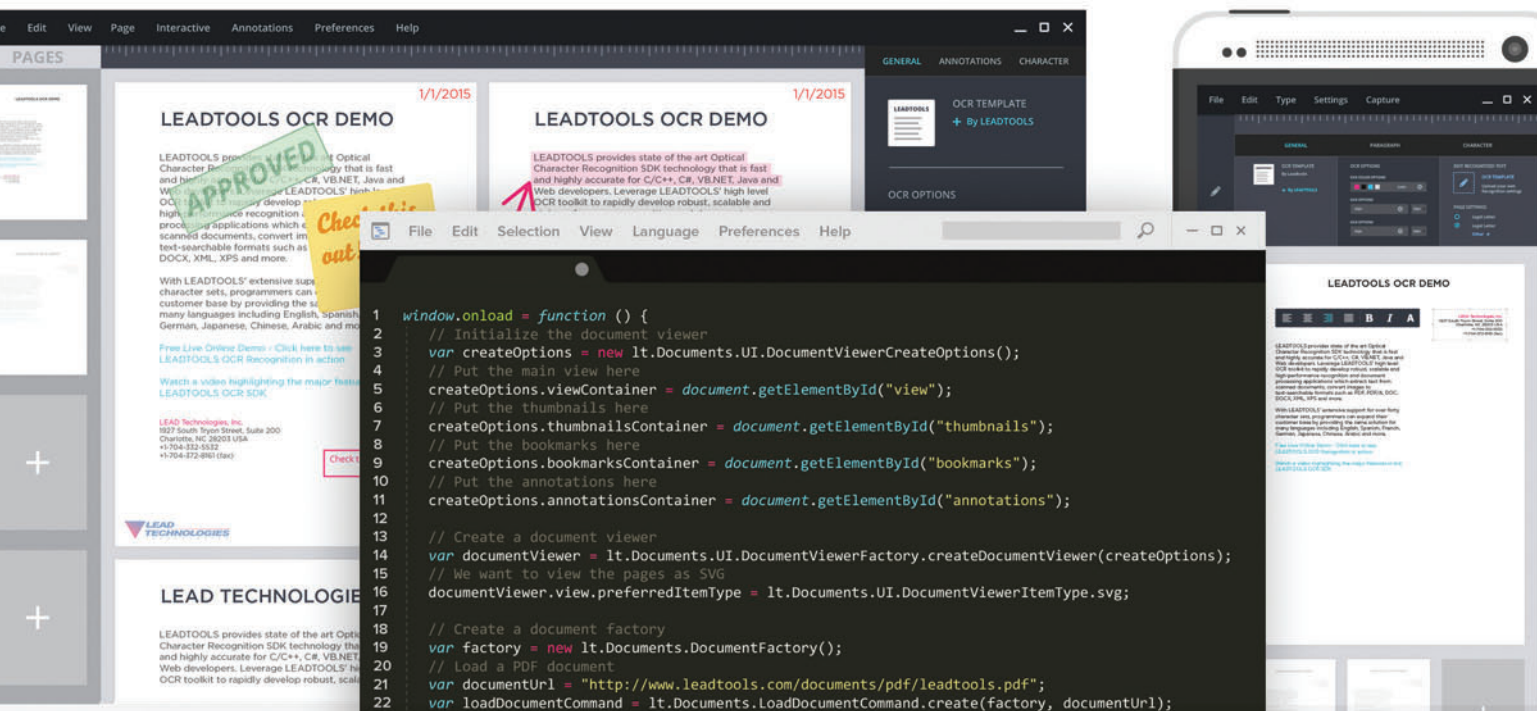
LEADTOOLS®

THE WORLD LEADER IN IMAGING SDKs

# DOCUMENT VIEWER & CONVERTER

## Create robust, fully-featured document viewing and editing solutions for any document, vector and raster file format *with only a few lines of code!*

- View document, vector and raster image formats in one viewer
- Document-specific interactive tools including Select Text, Search Text, Pan and Zoom
- Scalable Vector Graphics (SVG) display enables infinite zoom without loss of display quality or aliasing
- Easy to use and customizable programming interface available in .NET and HTML5/JavaScript
- Viewer seamlessly integrates with LEADTOOLS Document Converter to convert any file format into document formats including, PDF, PDF/A, DOX/DOCX, XLS/XLSX, PPT/PPTX, XPS, Text, RTF, HTML, MOBi, ePUB and more
- Perform document-to-document conversion at 100% accuracy without the need for OCR




DOWNLOAD OUR 60 DAY EVALUATION
**WWW.LEADTOOLS.COM**

LEAD TECHNOLOGIES INCORPORATED

SALES@LEADTOOLS.COM
**800.637.1840**



MICHAEL DESMOND

# Analyze This

In this month's issue of *MSDN Magazine*, Microsoft Senior Program Manager for the Managed Languages Team Alex Turner writes about the new Analyzer functionality in Visual Studio 2015 Preview and the .NET Compiler Platform (known previously by the code name "Roslyn"). His feature, "Adding a Code Fix to Your Roslyn Analyzer," is actually the second in a two-part exploration of the new Analyzer functionality. The first part, "Use Roslyn to Write a Live Code Analyzer for Your API" (msdn.microsoft.com/magazine/dn879356), appeared in last month's special issue on Visual Studio 2015 and Microsoft Azure.

> "Every issue your Analyzer detects is one less e-mail you'll get from the teams using your libraries, which helps your team save time and focus on development."
>
> *Alex Turner, Microsoft Senior Program Manager, Microsoft Managed Languages Team*

There's a reason we're running multiple articles on this topic. As Turner notes, the addition of diagnostic Analyzers to Visual Studio 2015 changes the game for developers, enabling real-time feedback about detected code issues as you type. What's more, Analyzers can provide custom guidance that's specific to the APIs being used. The opportunities to enable and enforce best practices across the coding environment are hugely compelling, and promise to help eliminate many common flaws and errors that plague code builds today. The Azure Code Analysis package is a case in point. The package provides a set of rules that detect code issues that can impair the scalability, reliability and security of cloud applications.

This is a huge step forward from traditional code analysis, which can only go to work on code at build time. As Turner tells it, the response from developers has been "amazing." "The community has already built open source Analyzer projects on GitHub such as Code Cracker, which now has over 40 diagnostic rules and over 300 commits. Not bad for a Visual Studio 2015 feature that's still in Preview," Turner says.

Helping spur early adoption is the decision by Microsoft to rely on NuGet for package management. Turner says he expects to see dev teams use NuGet not just to download external packages, but to enable private package servers to manage internal libraries. Authorized users can discover these libraries and distribute Analyzers with them.

"Every issue your Analyzer detects is one less e-mail you'll get from the teams using your libraries, which helps your team save time and focus on development," Turner says.

One misconception around Analyzers is that developers must be a compiler or language expert to write them. Turner recounts a recent boot camp session where he set attendees off to build Analyzers after a 90-minute talk.

"Within the first two hours, they'd already written 10 meaningful Analyzers, such as one that made sure sensitive information like connection strings or passwords don't end up in string literals by accident," he says. "Roslyn makes it easy for every dev team to write custom code analysis that enforces their own coding practices and business rules, well before problems make it as far as a code review."

For those getting started with the Analyzer functionality in Visual Studio 2015, Turner suggests that developers start simple. Implement a streamlined version of each rule and build it out in increments. The payback will happen quickly, he says, because developers can often catch 80 percent of the cases they hope to snare in less than 100 lines of code.

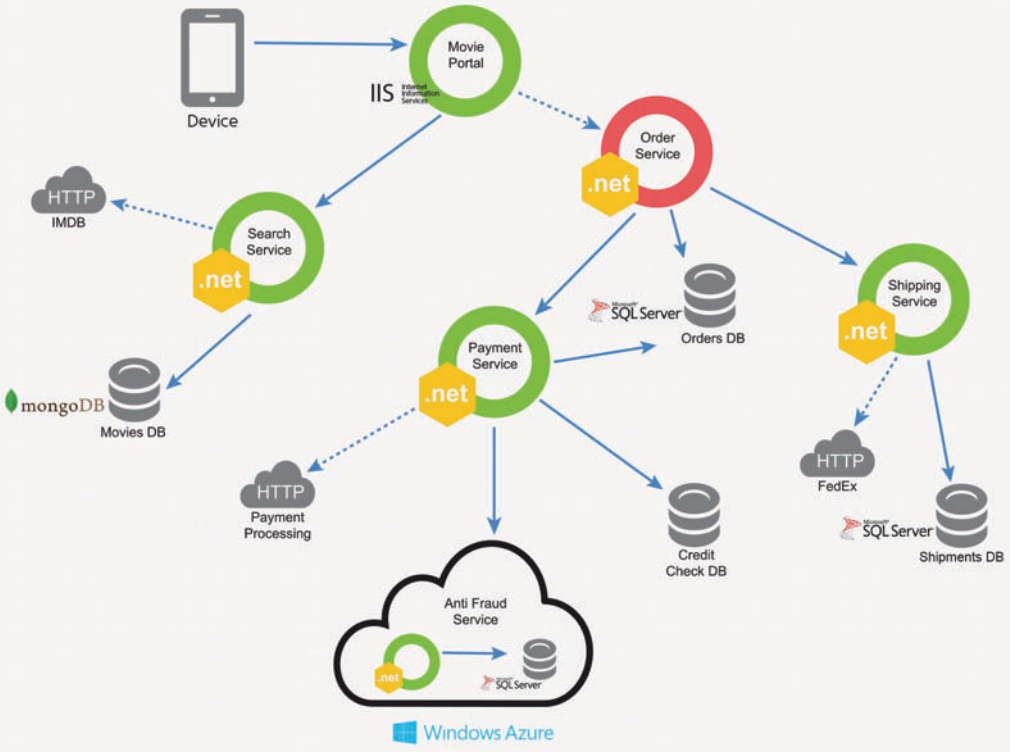
"Once you've got a successful Analyzer that's squiggling real problems, go ahead and dig in to start catching any special cases you find slipping through," Turner adds. "Analyzers don't need to be perfect to start saving your team lots of debugging time."

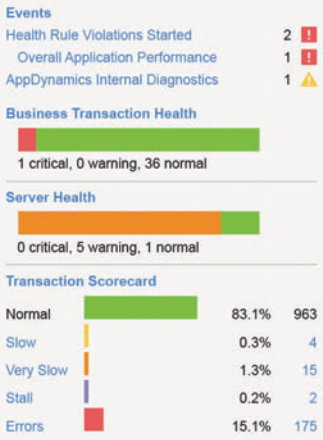
Visit us at msdn.microsoft.com/magazine. Questions, comments or suggestions for *MSDN Magazine*? Send them to the editor: mmeditor@microsoft.com.

APPDYNAMICS

# There's nothing about .NET that AppDynamics doesn't see.

AppDynamics gives you the visibility to take command of your .NET application's performance, no matter how complicated or distributed your environment is.




When your business runs on .NET and SQL Server, count on AppDynamics to give you the complete visibility you need to be sure they are delivering the performance and business results you need — no matter how complex, distributed or asynchronous your environment, live 'in production' or during development.

See every line of code. Get a complete view of your environment with deep code diagnostics and auto-discovery. Understand performance trends with dynamic baselining. And drastically reduce time to root cause and remediation.

See why the world's largest .NET deployments rely on the AppDynamics Application Intelligence Platform. Sign up for a FREE trial today at **www.appdynamics.com/Microsoft**.

appdynamics.com



DINO ESPOSITO

# Lightweight Client-Side Device Detection

There are two basic options for creating device-friendly Web sites—create a separate mobile site (m-site) or rework a site so it can intelligently adapt content and behavior to the current screen size and orientation. Neither of those two strategies is ideal in all cases. Therefore, the most insightful approach is the classic, "It depends," approach.

The m-site approach has two major drawbacks. First, all devices are different in terms of screen size and other physical capabilities (from smartphones, tablets and mini-tablets to older cell phones, phablets, wearable devices and so on). In the end, how do you actually define "mobile"?

Years ago, having an m-site made more sense because there was a clear separation between desktop computers and everything else. Today, the space of "everything else" has so many options that a generic unified mobile site is out of the question. The second drawback of an m-site is it requires users to navigate to a distinct URL, typically some m.yourserver.com. This is no longer a desirable solution and only acceptable as a temporary fix.

> The key question is how would you go about making a Web site responsive?

The other option—a responsive UI—isn't without its issues, either. The key question is how would you go about making a Web site responsive? A responsive site is commonly associated with the idea that it was built with responsive Web design (RWD). RWD is a development approach that uses the browser implementation of CSS Media Queries to detect screen size and orientation and lets you define breakpoints (essentially a pixel width) to automatically apply a different CSS.

The net effect is attractive. As you resize the browser window and trespass one of those visual breakpoints, the appearance of the site changes to make better use of the available real estate. The content adapts at no extra cost to any full-screen mobile browser used to visit the site. RWD is device-agnostic and scalable to any number and type of devices available now and in the future.

As pointed out in my December 2014 column, "Effective Image Handling in Responsive Web Sites" (msdn.microsoft.com/magazine/dn857356), being device-agnostic is both a strength and weakness of RWD. If your customers are happy with the results you can achieve through RWD (mostly performance and usability), then you're all set. RWD is your baby and you're done.

RWD usually works well with sites presenting content without requiring much interaction or wizard-like workflows. The more forms you have, the more you need users to select and type. In that case, the one-size-fits-all approach like RWD is less appropriate. So where are we, then?

The industry demands an effective way to serve appropriate content to any device without remembering a different URL. RWD is a popular approach to achieve this end. However, RWD is unapologetically device-agnostic. Developers don't love doing device detection. There are bad memories of when making a Web site look the same across multiple browsers required parsing the user agent string and bringing it to countless branches of code.

## Device Detection over Feature Detection

While you wait for the time when all browsers eventually expose platform and capabilities through a standard object model, parsing the user agent string is currently the only way to figure out which browser is actually requesting pages from a Web site. Parsing a user agent is a challenge, but there are a few tools that can mitigate the pain to some extent.

One of these tools is the script you get from detectmobilebrowsers.com. This script uses regular expressions to check a list of known mobile keywords and answer the question: "Is it mobile?" In this context, mobile simply means it's not a desktop browser. Another tool is Modernizr, which has a long list of plug-ins for user agent parsing and detecting touch events to try to detect a tablet.

Modernizr, however, isn't the right tool to detect anything other than JavaScript-detectable features. And whether the browser is a tablet or a smartphone isn't something you can detect with JavaScript. You can only ask the browser about its identity through JavaScript, but most browsers (especially mobile browsers) return inaccurate information for a number of reasons, including being incorrectly recognized by legacy code based on user agents.

Modernizr heralds the flag of feature detection versus device detection. It's an apples-to-oranges comparison. They're different things serving different purposes. If you need to determine the form factor of the requesting device, feature detection isn't the way. Some developers detect tablets and smartphones via JavaScript by asking Modernizr to check for touch events. This is increasingly unreliable considering the growing number of touch-enabled desktops and devices you want to treat as desktops.

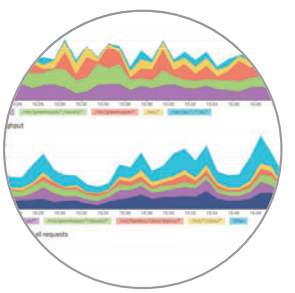
# One source of truth

See all your data. Boost performance. Drive accountability for everyone.



**Mobile Developers**
End-to-end visibility, 24/7 alerting, and crash analysis.


**Front-end Developers**
Deep insights into your browser-side app's engine.



**IT Operations**
Faster delivery.
Fewer bottlenecks.
More stability.

**App Owners**
Track engagement.
Pinpoint issues.
Optimize usability.

Move from finger-pointing blame to data-driven accountability.
Find the truth with a single source of data from multiple views.
**newrelic.com/truth**



New Relic.

You need expert help analyzing user agents and returning easily consumable information. Expert help means some framework that's continuously updated to add new devices as they hit the market. It also implies sophisticated parsing logic that properly handles false positive and uses statistical analysis to work around incorrect information passed by some mobile browsers. Frankly, it's a lot of work. There are some companies doing it for you, though. These tools are called Device Description Repositories (DDRs).

## The WURFL.JS Endpoint

A lightweight form of JavaScript device detection can improve the UX in client-side-intensive Web applications, such as a single-page application (SPA). WURFL (wurfl.sourceforge.net) is a popular DDR that has been around for a few years as a server-side solution. I wrote about using WURFL in an ASP.NET MVC application in my August 2013 column, "Creating Mobile-Optimized Views in ASP.NET MVC 4, Part 2: Using WURFL" (msdn.microsoft.com/magazine/dn342866.aspx).

> WURFL is a popular DDR that has been around for a few years as a server-side solution.

Server-side WURFL is subject to a license fee whether you use it on-premises or access the DDR in the cloud. The WURFL team recently released a free HTTP endpoint (WURFL.JS) that you can invoke from the client side through JavaScript. To enable JavaScript device detection via the WURFL.JS endpoint, all you need to do is add the following line to HTML (in ASP.NET, you can even place it in the master page or layout file):

```
<script type="text/javascript" src="http://wurfl.io/wurfl.js"></script>
```

The referenced resource—wurfl.js—is not clearly a plain JavaScript file you can download and host on-premises or upload in your cloud site. It's a JavaScript-like HTTP endpoint that injects a JavaScript object right in the Document Object Model (DOM). The net effect is once the browser has made a call to the endpoint, your DOM contains the following:

```
var WURFL = {
  "complete_device_name":"iPhone 5",
  "is_mobile":false,
  "form_factor":"Smartphone"};
```

The browser sends its user agent string while making the request. Backed by the server-side WURFL framework, the endpoint analyzes the string and determines key information about the requesting device. Such information is then formatted into three properties in a global object named WURFL (see **Figure 1**). It's interesting to notice WURFL.JS can also reliably detect whether your Web page is being viewed from within the WebView component of a native mobile app. This happens when the form_factor property returns App.

Figure 1 **Device Information WURFL.JS Injects in the DOM Page**

| Property | Description |
|---|---|
| complete_device_name | Contains a descriptive name for the detected device that typically includes vendor and device name (for example, iPhone 5). |
| form_factor | Contains one of a few predefined strings such as: Desktop, App, Tablet, Smartphone, Feature Phone, Smart-TV, Robot, Other non-Mobile, Other Mobile. |
| is_mobile | Boolean value, indicates whether the device is not a desktop. |

WURFL.JS uses a lot of caching to ensure a quick response. In the development phase, you can switch off the cache by adding debug=true to the URL. When the DOM ready event is fired, you can safely use the WURFL object to enable or disable client-side features or request optional data to the server.

## WURFL.JS Adds Power to RWD

Suppose you have a page with video and you don't want it to play on smartphones for performance reasons. With plain RWD, you can't make adjustments like that. If you hide the video player below a given breakpoint, you won't be able to play anything when the desktop browser is resized to a tiny window. WURFL.JS used with RWD solves the issue, as shown here:

```
<script type="text/javascript">
  $(document).ready(function () {
    if (WURFL.form_factor == "Smartphone") {
      $("#video_player").hide();
    }
  });
</script>
```

First, load the page and style it according to the RWD layout. Next, use WURFL to check the form factor and hide the video player. When a desktop browser is resized to the size of a smartphone, users will still be able to play videos, but not when it's a real smartphone. An even better formulation of the earlier code is shown here:

```
<script type="text/javascript">
  $(document).ready(function () {
    if (WURFL.form_factor != "Desktop" &&
        WURFL.form_factor != "Tablet") {
      $("#video_player").hide();
    }
  });
</script>
```

In this case, the video player is hidden in all cases, except when the device is a desktop or tablet. There's another more interesting scenario for WURFL.JS. Suppose you have an RWD site in production where all views are generated on the server, within an ASP.NET MVC application, for example.

At some point, you'll get feedback that smartphone users aren't having an appropriate experience. Should you consider creating an entirely new m-site for them? If you can use the server-side services of WURFL, then you can easily implement a view switcher within the same site, as demonstrated in my August 2013 column.

This approach would save most of the work you've done. Otherwise, there's not much you can do other than create a separate site and implement some redirect mechanism, as demonstrated in my January 2015 column, "Mobilize an Existing Web Site" (msdn.microsoft.com/magazine/dn890366). To display regular and m-site under the same URL, you still need some good device detection done on the server side.

A pleasant side effect of using WURFL.JS is you can collect device information from the WURFL object and pass it on to Google Analytics. That gives you an immediate measure of how users are visiting your site and which device they use most. If you upgraded to the new analytics.js you need the following:

```
<script type="text/javascript">
  /* Universal tracking code of Google Analytics:
    see http://goo.gl/HakYmP
  */

  ga('create', 'UA-XXXX-Y', 'auto');
  ga('send', 'pageview', {'dimension1': WURFL.form_factor});
</script>
```

If you're using the classic Google Analytics script (ga.js), here's the slight change you need to implement:

```
_gaq.push(['_setCustomVar', 1, 'form_factor', WURFL.form_factor, 1]);
```

Google Analytics has a bunch of built-in features to track mobile and tablet traffic. The mobile traffic incorporates tablet traffic, as well, and can't immediately separate smartphone from tablet, for example. WURFL.JS adds handy information missing at first in Google Analytics so you can create custom reports just focusing on the numbers instead of data projection. WURFL.JS, though, is just a data provider. It works with Google Analytics, but you can also use it with other analytics tools.

## Wrapping Up

In many cases, an RWD site is harder (more expensive) to implement than a plain ASP.NET Web Forms Web site. Don't listen to the sirens of RWD. RWD is a great solution for desktops and high-end devices, but it might not be appropriate when effectively implementing functions on small devices is crucial for the business. RWD makes a point of being device-agnostic. This means RWD Web sites serve the same content to a 480x360 resized desktop browser window and a full-screen small feature phone.

Hardware and connectivity may be significantly different in the two cases. Performance is objectively a sore point of RWD. At the same time, it might be not be painful in the same way for all sites. Performance issues affect primarily low-level devices. If those devices aren't common site visitors, you can go with RWD and be happy.

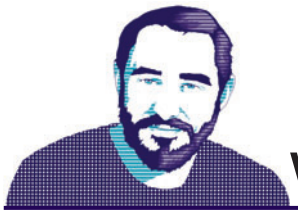
However, the amount of data being served and user expectations may grow in the near future. This would render more devices inadequate and, ideally, requiring an ad hoc view. In this article, I've presented a lightweight and nearly unobtrusive client-side solution for detecting the underlying device—WURFL.JS.

WURFL.JS is an endpoint that injects device information in the DOM, specifically the form factor. In other words, it tells you whether the device is a desktop, a tablet, a smartphone, an old phone, an Xbox or perhaps a native app. Based on that, you can arrange small changes in the pages and even feed analytics tools with form factor information.

Knowing how your site performs on a per-form-factor basis is a powerful indicator of its success and the areas you need to improve. If you want more details on how to use WURFL.JS with Google Analytics, check bit.ly/1u0lpGB. ■

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

**Thanks** to the following technical expert for reviewing this article: *Jon Arne Saeteras*





KENNY KERR

# COM Smart Pointers Revisited

After the second coming of COM, otherwise known as the Windows Runtime, the need for an efficient and reliable smart pointer for COM interfaces is more important than ever. But what makes for a good COM smart pointer? The ATL CComPtr class template has been the de facto COM smart pointer for what feels like decades. The Windows SDK for Windows 8 introduced the ComPtr class template as part of the Windows Runtime C++ Template Library (WRL), which some hailed as a modern replacement for the ATL CComPtr. At first, I also thought this was a good step forward, but after a lot of experience using the WRL ComPtr, I've come to the conclusion it should be avoided. Why? Keep reading.

So what should be done? Should we return to ATL? By no means, but perhaps it's time to apply some of the modern C++ offered by Visual C++ 2015 to the design of a new smart pointer for COM interfaces. In the Connect(); Visual Studio 2015 & Microsoft Azure Special Issue, I showed how to make the most of Visual C++ 2015 to easily implement IUnknown and IInspectable using the Implements class template. Now I'm going to show you how to use more of Visual C++ 2015 to implement a new ComPtr class template.

> I actually like the COM approach to reference counting, but I want a library to take care of it for me.

Smart pointers are notoriously difficult to write, but thanks to C++11, it's not nearly as difficult as it once was. Part of the reason for this has to do with all of the clever tricks library developers devised to work around the lack of expressiveness in the C++ language and standard libraries, in order to make their own objects act like built-in pointers while remaining efficient and correct. In particular, rvalue references go a long way toward making life so much easier for us library developers. Another part is simply hindsight—seeing how existing designs have fared. And, of course, there's every developer's dilemma: showing restraint and not trying to pack every conceivable feature into a particular abstraction.

At the most basic level, a COM smart pointer must provide resource management for the underlying COM interface pointer. This implies that the smart pointer will be a class template and store an interface pointer of the desired type. Technically, it doesn't actually need to store an interface pointer of a particular type, but could instead just store an IUnknown interface pointer, but then the smart pointer would have to rely on a static_cast whenever the smart pointer is dereferenced. This can be useful and conceptually dangerous, but I'll talk about it in a future column. For now, I'll begin with a basic class template for storing a strongly typed pointer:

```
template <typename Interface>
class ComPtr
{
public:

  ComPtr() noexcept = default;

private:

  Interface * m_ptr = nullptr;
};
```

Longtime C++ developers might wonder at first what this is all about, but chances are that most active C++ developers won't be too surprised. The m_ptr member variable relies on a great new feature that allows non-static data members to be initialized where they're declared. This dramatically reduces the risk of accidentally forgetting to initialize member variables as constructors are added and changed over time. Any initialization explicitly provided by a particular constructor takes precedence over this in-place initialization, but for the most part this means that constructors need not worry about setting such member variables that would otherwise have started off with unpredictable values.

Given the interface pointer is now assured to be initialized, I can also rely on another new feature to explicitly request a default definition of special member functions. In the previous example, I'm requesting the default definition of the default constructor—a default default constructor, if you will. Don't shoot the messenger. Still, the ability to default or delete special member functions along with the ability to initialize member variables at the point of declaration are among my favorite features offered by Visual C++ 2015. It's the little things that count.

The most important service a COM smart pointer must offer is that of shielding the developer from the perils of the intrusive COM reference-counting model. I actually like the COM approach to reference counting, but I want a library to take care of it for me. This surfaces in a number of subtle places throughout the ComPtr class template, but perhaps the most obvious is when a caller dereferences the smart pointer. I don't want a caller to write something like what follows, accidentally or otherwise:

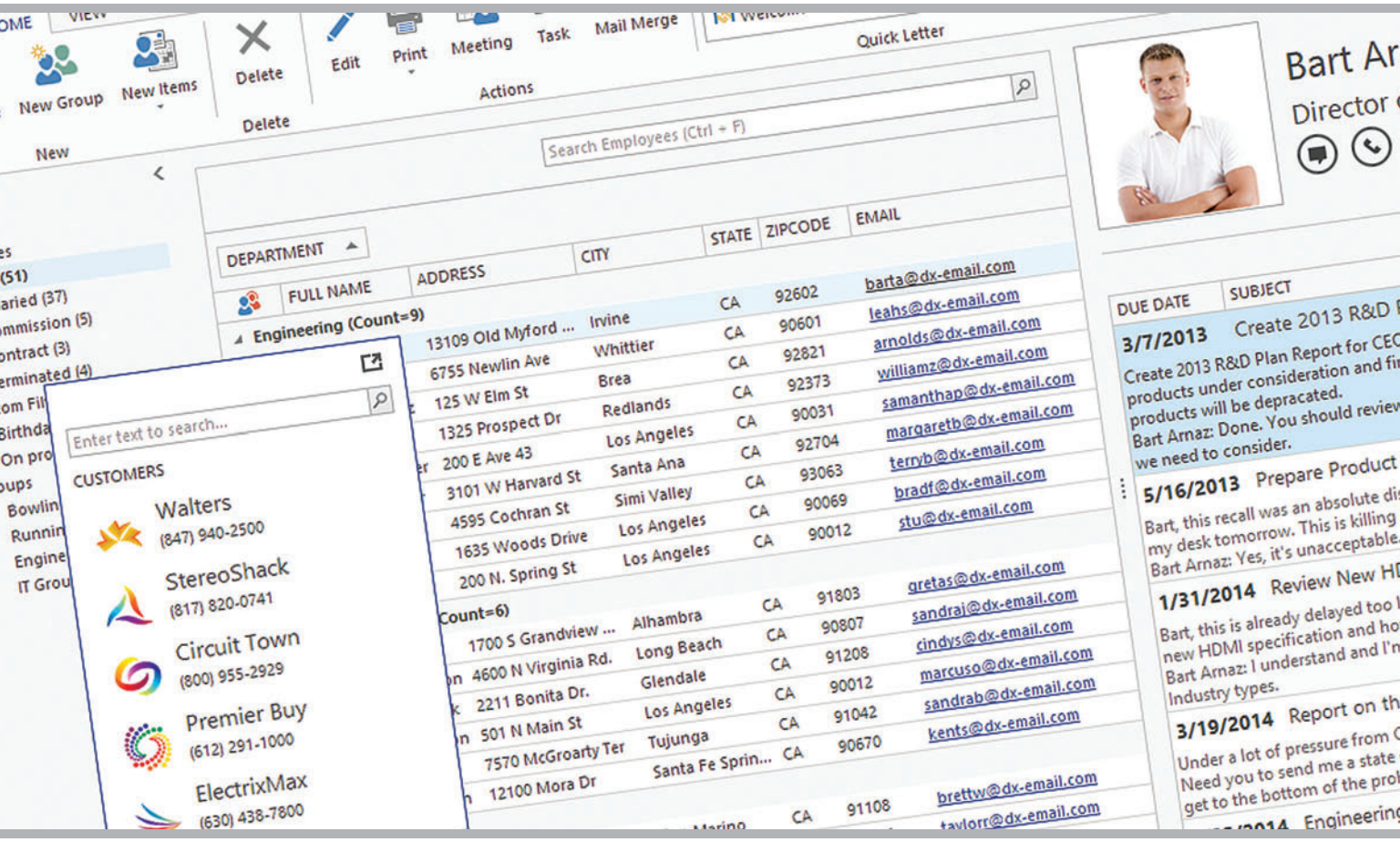
```
ComPtr<IHen> hen;

hen->AddRef();
```

DevExpress®

# Your Next Great App Starts Here

With DevExpress tools, you'll create high-performance, high-impact .NET solutions that fully replicate the look, feel and user-experience of Microsoft Office®




## Unleash the UI Superhero in You

Download your free 30-day trial today
devexpress.com/try

WIN ASP MVC WPF HTML

The ability to call the AddRef or Release virtual functions should be exclusively under the purview of the smart pointer. Of course, the smart pointer must still allow the remaining methods to be called via such a dereferencing operation. Normally, a smart pointer's dereference operator might look something like this:

```
Interface * operator->() const noexcept
{
  return m_ptr;
}
```

That works for COM interface pointers and there's no need for an assertion because an access violation is more instructive. But this implementation will still allow a caller to call AddRef and Release. The solution is simply to return a type that prohibits AddRef and Release from being called. A little class template comes in handy:

```
template <typename Interface>
class RemoveAddRefRelease : public Interface
{
  ULONG __stdcall AddRef();
  ULONG __stdcall Release();
};
```

The RemoveAddRefRelease class template inherits all of the template argument's methods, but declares AddRef and Release private so that a caller may not accidentally refer to those methods. The smart pointer's dereference operator can simply use static_cast to protect the returned interface pointer:

```
RemoveAddRefRelease<Interface> * operator->() const noexcept
{
  return static_cast<RemoveAddRefRelease<Interface> *>(m_ptr);
}
```

This is just one example where my ComPtr deviates from the WRL approach. WRL opts to make all of IUnknown's methods private, including QueryInterface, and I see no reason for restricting callers in that way. It means that WRL must inevitably provide alternatives for this essential service and that leads to added complexity and confusion for callers.

Because my ComPtr decidedly takes command of reference counting, it had better do so correctly. Well, I'll start with a pair of private helper functions beginning with one for AddRef:

```
void InternalAddRef() const noexcept
{
  if (m_ptr)
  {
    m_ptr->AddRef();
  }
}
```

This isn't all that exciting, but there are a variety of functions that require a reference to be taken conditionally and this will ensure I do the right thing every time. The corresponding helper function for Release is a bit more subtle:

```
void InternalRelease() noexcept
{
  Interface * temp = m_ptr;

  if (temp)
  {
    m_ptr = nullptr;
    temp->Release();
  }
}
```

Why the temporary? Well, consider the more intuitive but incorrect implementation that roughly mirrors what I did (correctly) inside the InternalAddRef function:

```
if (m_ptr)
{
  m_ptr->Release(); // BUG!
  m_ptr = nullptr;
}
```

The problem here is that calling the Release method might set off a chain of events that could see the object being released a second time. This second trip through InternalRelease would again find a non-null interface pointer and attempt to Release it again. This is admittedly an uncommon scenario, but the job of the library developer is to consider such things. The original implementation involving a temporary avoids this double Release by first detaching the interface pointer from the smart pointer and only then calling Release. Looking through the annals of history, it appears as if Jim Springfield was the first to catch this vexing bug in ATL. Anyway, with these two helper functions in hand, I can begin to implement some of the special member functions that help to make the resulting object act and feel like a built-in object. The copy constructor is a simple example.

Unlike smart pointers that provide exclusive ownership, copy construction should be allowed for COM smart pointers. Care must be taken to avoid copies at all costs, but if a caller really wants a copy then a copy is what it gets. Here's a simple copy constructor:

```
ComPtr(ComPtr const & other) noexcept :
  m_ptr(other.m_ptr)
{
  InternalAddRef();
}
```

This takes care of the obvious case of copy construction. It copies the interface pointer before calling the InternalAddRef helper. If I left it there, copying a ComPtr would feel mostly like a built-in pointer, but not entirely so. I could, for example, create a copy like this:

```
ComPtr<IHen> hen;
ComPtr<IHen> another = hen;
```

This mirrors what I can do with raw pointers:

```
IHen * hen = nullptr;
IHen * another = hen;
```

But raw pointers also permit this:

```
IUnknown * unknown = hen;
```

With my simple copy constructor, I'm not permitted to do the same thing with ComPtr:

```
ComPtr<IUnknown> unknown = hen;
```

Even though IHen must ultimately derive from IUnknown, ComPtr<IHen> doesn't derive from ComPtr<IUnknown> and the compiler considers them unrelated types. What I need is a constructor that acts as a logical copy constructor for other logically related ComPtr objects—specifically, any ComPtr with a template argument that's convertible to the constructed ComPtr's template argument. Here, WRL relies on type traits, but this isn't actually necessary. All I need is a function template to provide for the possibility of conversion and then I'll simply let the compiler check whether it's actually convertible:

```
template <typename T>
ComPtr(ComPtr<T> const & other) noexcept :
  m_ptr(other.m_ptr)
{
  InternalAddRef();
}
```

It's when the other pointer is used to initialize the object's interface pointer that the compiler checks whether the copy is actually meaningful. So this will compile:

```
ComPtr<IHen> hen;
ComPtr<IUnknown> unknown = hen;
```

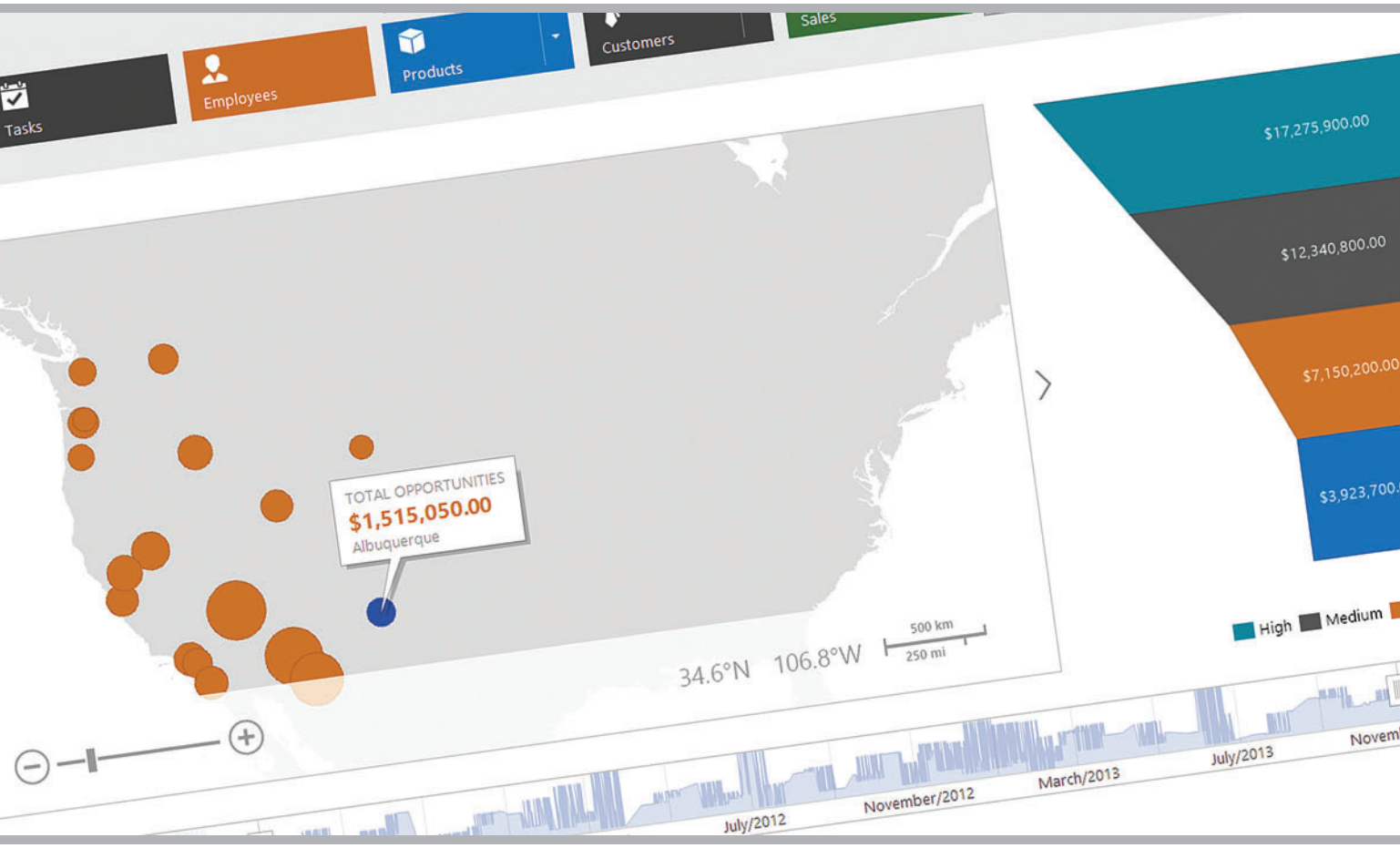
But this won't:

```
ComPtr<IUnknown> unknown;
ComPtr<IHen> hen = unknown;
```

DevExpress®

# Yeah, You Can Touch This

With DevExpress tools, you'll deliver elegant, touch-first apps that are built for today and ready for tomorrow




## Unleash the UI Superhero in You

Download your free 30-day trial today
devexpress.com/try

WIN ASP MVC WPF

And that's as it should be. Of course, the compiler still considers the two very much different types, so the constructor template won't actually have access to the other's private member variable, unless I make them friends:

```
template <typename T>
friend class ComPtr;
```

You might be tempted to remove some of the redundant code because IHen is convertible to IHen. Why not just remove the actual copy constructor? The problem is that this second constructor isn't considered a copy constructor by the compiler. If you omit the copy constructor, the compiler will assume you meant to remove it and object to any reference to this deleted function. Onward.

With copy construction taken care of, it's very important that ComPtr also provide move construction. If a move is permissible in a given scenario, ComPtr should allow the compiler to opt for that as it will save a reference bump, which is far more costly in comparison to a move operation. A move constructor is even simpler than the copy constructor because there's no need to call InternalAddRef:

```
ComPtr(ComPtr && other) noexcept :
  m_ptr(other.m_ptr)
{
  other.m_ptr = nullptr;
}
```

It copies the interface pointer before clearing or resetting the pointer in the rvalue reference, or the object being moved from. In this case, however, the compiler is not so picky and you can simply eschew this move constructor for a generic version that supports convertible types:

```
template <typename T>
ComPtr(ComPtr<T> && other) noexcept :
  m_ptr(other.m_ptr)
{
  other.m_ptr = nullptr;
}
```

And that wraps up the ComPtr constructors. The destructor is predictably simple:

```
~ComPtr() noexcept
{
  InternalRelease();
}
```

I've already taken care of the nuances of destruction inside the InternalRelease helper, so here I can simply reuse that goodness. I've discussed copy and move construction, but the corresponding assignment operators must also be provided for this smart pointer to feel like a real pointer. In order to support those operations, I'm going to add another pair of private helper functions. The first is for safely acquiring a copy of a given interface pointer:

```
void InternalCopy(Interface * other) noexcept
{
  if (m_ptr != other)
  {
    InternalRelease();
    m_ptr = other;
    InternalAddRef();
  }
}
```

Assuming the interface pointers are not equal (or not both null pointers), the function releases any existing reference before taking a copy of the pointer and securing a reference to the new interface pointer. In this way, I can easily call InternalCopy to take ownership of a unique reference to the given interface even if the smart pointer already holds a reference. Similarly, the second helper deals with safely moving a given interface pointer, along with the reference count it represents:

```
template <typename T>
void InternalMove(ComPtr<T> & other) noexcept
{
  if (m_ptr != other.m_ptr)
  {
    InternalRelease();
    m_ptr = other.m_ptr;
    other.m_ptr = nullptr;
  }
}
```

While InternalCopy naturally supports convertible types, this function is a template to provide this capability for the class template. Otherwise, InternalMove is largely the same, but logically moves the interface pointer rather than acquiring an additional reference. With that out of the way, I can implement the assignment operators quite simply. First, the copy assignment, and as with the copy constructor, I must provide the canonical form:

```
ComPtr & operator=(ComPtr const & other) noexcept
{
  InternalCopy(other.m_ptr);
  return *this;
}
```

I can then provide a template for convertible types:

```
template <typename T>
ComPtr & operator=(ComPtr<T> const & other) noexcept
{
  InternalCopy(other.m_ptr);
  return *this;
}
```

But like the move constructor, I can simply provide a single generic version of move assignment:

```
template <typename T>
ComPtr & operator=(ComPtr<T> && other) noexcept
{
  InternalMove(other);
  return *this;
}
```

While move semantics are often superior to copy when it comes to reference-counted smart pointers, moves aren't without cost, and a great way to avoid moves in some key scenarios is to provide swap semantics. Many container types will favor swap operations to moves, which can avoid the construction of a tremendous load of temporary objects. Implementing swap functionality for ComPtr is quite straightforward:

```
void Swap(ComPtr & other) noexcept
{
  Interface * temp = m_ptr;
  m_ptr = other.m_ptr;
  other.m_ptr = temp;
}
```

I'd use the Standard swap algorithm but, at least in the Visual C++ implementation, the required <utility> header also indirectly includes <stdio.h> and I don't really want to force developers into including all of that just for swap. Of course, for generic algorithms to find my Swap method, I need to also provide a non-member (lowercase) swap function:

```
template <typename Interface>
void swap(ComPtr<Interface> & left, ComPtr<Interface> & right) noexcept
{
  left.Swap(right);
}
```

As long as this is defined in the same namespace as the ComPtr class template, the compiler will happily allow generic algorithms to make use of the swap.

Another nice feature of C++11 is that of explicit conversion operators. Historically, it took some messy hacks to produce a reliable, explicit Boolean operator for checking whether a smart pointer was logically not null. Today, it's as simple as this:

```
explicit operator bool() const noexcept
{
  return nullptr != m_ptr;
}
```

And that takes care of the special and practically special members that make my smart pointer behave much like a built-in type with as much assistance as I can possibly provide to help the compiler optimize any overhead away. What remains is a small selection of helpers that are necessary for COM applications in many cases. This is where care should be taken to avoid adding too many bells and whistles. Still, there are a handful of functions on which almost any nontrivial application or component will rely. First, it needs a way to explicitly release the underlying reference. That's easy enough:

```
void Reset() noexcept
{
  InternalRelease();
}
```

And then it needs a way to get the underlying pointer, should the caller need to pass it as an argument to some other function:

```
Interface * Get() const noexcept
{
  return m_ptr;
}
```

I might need to detach the reference, perhaps to return it to a caller:

```
Interface * Detach() noexcept
{
  Interface * temp = m_ptr;
  m_ptr = nullptr;
  return temp;
}
```

I might need to make a copy of an existing pointer. This might be a reference held by the caller that I'd like to hold on to:

```
void Copy(Interface * other) noexcept
{
  InternalCopy(other);
}
```

Or I might have a raw pointer that owns a reference to its target that I'd like to attach without an additional reference being procured. This can also be useful for coalescing references in rare cases:

```
void Attach(Interface * other) noexcept
{
  InternalRelease();
  m_ptr = other;
}
```

The final few functions play a particularly critical role, so I'll spend a few more moments on them. COM methods traditionally return references as out parameters via a pointer to a pointer. It's important that any COM smart pointer provide a way to directly capture such references. For that I provide the GetAddressOf method:

```
Interface ** GetAddressOf() noexcept
{
  ASSERT(m_ptr == nullptr);
  return &m_ptr;
}
```

This is again where my ComPtr departs from the WRL implementation in a subtle but very critical way. Notice that GetAddressOf asserts that it doesn't hold a reference before returning its address. This is vitally important, otherwise the called function will simply overwrite whatever reference may have been held and you've got yourself a reference leak. Without the assertion, such bugs are much harder to detect. On the other end of the spectrum is the ability to hand out references, either of the same type or for other interfaces the underlying object might implement. If another reference to the same interface is desired, I can avoid calling QueryInterface and simply return an additional reference using the convention prescribed by COM:

```
void CopyTo(Interface ** other) const noexcept
{
  InternalAddRef();
  *other = m_ptr;
}
```

And you might use it as follows:

```
hen.CopyTo(copy.GetAddressOf());
```

Otherwise, QueryInterface itself can be employed with no further help from ComPtr:

```
HRESULT hr = hen->QueryInterface(other.GetAddressOf());
```

This actually relies on a function template provided directly by IUnknown to avoid having to explicitly provide the interface's GUID.

COM methods traditionally return references as out parameters via a pointer to a pointer. It's important that any COM smart pointer provide a way to directly capture such references.

Finally, there are often cases where an app or component needs to query for an interface without necessarily passing it back to the caller in the classic COM convention. In those cases, it makes more sense to return this new interface pointer tucked snugly inside another ComPtr, as follows:

```
template <typename T>
ComPtr<T> As() const noexcept
{
  ComPtr<T> temp;
  m_ptr->QueryInterface(temp.GetAddressOf());
  return temp;
}
```

I can then simply use the explicit operator bool to check whether the query succeeded. Finally, ComPtr also provides all of the expected non-member comparison operators for convenience and to support various containers and generic algorithms. Again, this just helps to make the smart pointer act and feel more like a built-in pointer, all the while providing the essential services to properly manage the resource and provide the necessary services that COM apps and components expect. The ComPtr class template is just another example from Modern C++ for the Windows Runtime (moderncpp.com). ■

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

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

BEST SELLER

## ComponentOne Studio Enterprise 2014 v3 | 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
- A line of HTML5 and JavaScript products for enterprise application development
- Built in themes and an array of designers for creating custom themes and styling
- 40+ Windows 8.1 & Windows Phone 8.1 controls and Universal Windows app support
- All Microsoft platforms supported, Visual Studio 2013, ASP.NET, WinForms, WPF & more

BEST SELLER

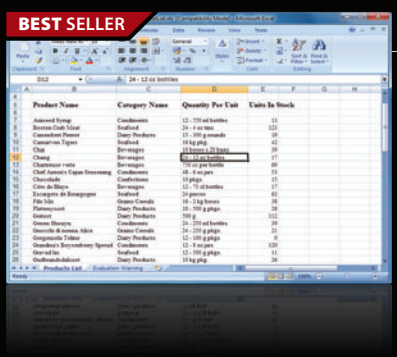
## Help & Manual Professional | from $583.10

ec software

**Easily create documentation for Windows, the Web and iPad.**

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

BEST SELLER



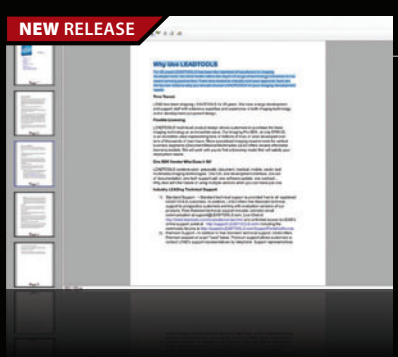
## Aspose.Total for .NET | from $2,449.02

ASPOSE
Your File Format APIs

**Every Aspose .NET component in one package.**

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

NEW RELEASE



## LEADTOOLS PDF Pro SDK V19 | from $1,495.00 SRP

LEAD TECHNOLOGIES

**Add powerful PDF read, write, view & editing functionality to desktop, tablet & mobile applications.**

- View PDF files as raster in .NET, WinRT and C/C++
- Merge, Split, Convert, Linearize and Distill PDF files
- Read and Extract text, hyperlinks, metadata and more from PDF
- Create the smallest file possible utilizing advanced PDF Optimizer
- Convert PDF to an image format such as JPEG or TIFF



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

# Share UI Code Across Mobile Platforms with Xamarin.Forms

Jason Smith

With Xamarin, you can use C# to build beautiful native mobile apps and share most of your code between platforms. Traditionally, you had to design a separate UI for each targeted platform. But with Xamarin.Forms, you can build one UI that renders natively across all of them.

Xamarin.Forms is a cross-platform UI abstraction layer. You can use it to share UI and back-end code between platforms and yet still deliver a fully native UI experience. Because they're native, your controls and widgets have the look and feel of each target platform.

Xamarin.Forms is fully compatible with the Model-View-ViewModel (MVVM) design pattern so you can bind page elements to properties and commands in a view model class.

If you prefer to design your pages declaratively, you can use XAML, a markup language with features such as resource dictionaries, dynamic resources, data binding, commands, triggers and behaviors.

This article discusses:

- Setting up a simple Xamarin.Forms project
- Using platform-specific code when needed
- The building blocks of Xamarin.Forms
- Showing data in a list view
- Navigating app pages
- Using animation

Technologies discussed:

Xamarin.Forms

Xamarin.Forms has a small, easy-to-use API. If you need deeper access to the native UI of a platform, you can create custom views and platform-specific renderers. This sounds complicated, but it's really just a way to get to native UIs, and there are plenty of samples on the Xamarin Web site to help you do it.

You can start by using any of the readymade pages, layouts and views that come with Xamarin.Forms. As your app matures and you discover new use cases and design opportunities, you might come to rely on Xamarin.Forms support for XAML, MVVM, custom platform-specific renderers, and a variety of other features such as animations and data templates.

I'll provide an overview of the Xamarin functionality with specific examples to illustrate how to share the bulk of your UI code across different target platforms and incorporate platform-specific code when needed.

## Getting Started

First, open the NuGet package manager in Visual Studio or Xamarin Studio and check for new releases of Xamarin.Forms. Because you won't be notified about new releases just by opening a Xamarin.Forms solution in either IDE, checking for updated NuGet packages is the only way to ensure you get the latest enhancements.

**Create a Xamarin.Forms Solution** When you're sure you've got the latest version of Xamarin.Forms, create a Blank App (Xamarin.Forms Portable) solution.

Your solution has three platform-specific projects and one Portable Class Library (PCL). Create your pages in the PCL. Start by creating a basic login page.

Figure 1 **Adding Views (Controls)**

```
public class LogInPage : ContentPage
{
  public LogInPage()
  {
    Entry userEntry = new Entry { Placeholder = "Username" };
    Entry passEntry =
      new Entry { Placeholder = "Password", IsPassword = true };

    Button submit = new Button { };
    Content = new StackLayout
    {
      Padding = 20,
      VerticalOptions = LayoutOptions.Center,
      Children = { userEntry, passEntry, submit }
    };
  }
}
```

**Use C# to Create a Page** Add a class to your PCL project. Then add controls (called "views" in Xamarin), as shown in **Figure 1**.

To show the page when an app starts, open the MyApp class and assign an instance of it to the MainPage property:

```
public class MyApp : Application
{
  public MyApp()
  {
    MainPage = new LogInPage();
  }
}
```

This is a good time to discuss the Application class. Starting in v1.3.0, all Xamarin.Forms apps will contain this class. It's the entry point of a Xamarin.Forms app and, among other things, it provides lifecycle events, as well as a persistent data store (the Properties dictionary) for any serializable data. If you have to get to an instance of this class from anywhere in your app, you can use the static Application.Current property.

In the preceding example, I removed the default code inside of the application class and replaced it with a single line of code that makes the LogInPage appear when you run the app. It appears when the app runs because this code assigns a page (the LogInPage) to the MainPage property of the Application class. You have to set it in the constructor of the Application class.

You can also override three methods in this class:

- The OnStart method, which is called when an app is first started.
- The OnSleep method, which is called when the app is about to go into a background state.
- The OnResume method, which is called when an app returns from a background state.

In general, pages aren't very interesting until you connect them with some sort of data or behavior, so I'll show how to do that.

**Bind a Page to Data** If you use the MVVM design pattern, you might create a class such as the one in **Figure 2** that implements the INotifyPropertyChanged interface.

You can bind the views of your login page to the properties of that class, as shown in **Figure 3**.

To read more about how to bind to data in your Xamarin.Forms app, see "From Data Bindings to MVVM" on the Xamarin site at bit.ly/1uMolUX.

**Use XAML to Create a Page** For smaller apps, creating your UIs by using C# is a perfectly reasonable approach. However, as the size of your app grows, you might find yourself typing a lot of repetitive code. You can avoid that problem by using XAML instead of C# code.

Add a Forms XAML Page item to your PCL project. Then add markup to the page, as shown in **Figure 4**.

The XAML in **Figure 4** might look familiar if you've written Windows Presentation Foundation (WPF) apps. However, the tags are different because they refer to Xamarin.Forms types. Also, the root element refers a subclass of the Xamarin.Forms.Element class. All XAML files in a Xamarin.Forms app must do this.

To read more about using XAML to create pages in a Xamarin.Forms app, see "XAML for Xamarin.Forms" on the Xamarin Web site at bit.ly/1xAKvRN.

Figure 2 **Implementing the INotifyPropertyChanged Interface**

```
public class LoginViewModel : INotifyPropertyChanged
{
  private string usrnmTxt;
  private string passWrd;

  public string UsernameText
  {
    get { return usrnmTxt; }
    set
    {
      if (usrnmTxt == value)
        return;
      usrnmTxt = value;
      OnPropertyChanged("UsernameText");
    }
  }

  public string PassWordText
  {
    get { return passWrd; }
    set
    {
      if (passWrd == value)
        return;
      passWrd = value;
      OnPropertyChanged("PassWrd");
    }
  }

  public event PropertyChangedEventHandler PropertyChanged;

  private void OnPropertyChanged(string propertyName)
  {
    if (PropertyChanged != null)
    {
      PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
    }
  }
}
```

## Design for a Specific Platform

Xamarin.Forms has a relatively small number of APIs compared with other native mobile platforms. That makes it easier for you to design your pages, but sometimes Xamarin.Forms doesn't render a view exactly the way you want it to on one or more of your platform targets.

If you run up against this barrier, just create a custom view, which is just a subclass of any view that's available in Xamarin.Forms.

To make the view appear on a page, extend the view renderer. In more advanced cases, you can even create a custom renderer from scratch. Custom renderer code is specific to a platform, so you can't share it. But this approach could be worth the cost in order to introduce native features and usability to the app.

**Create a Custom View** First, create a subclass of any view that's available in Xamarin.Forms. Here's code for two custom views:

```
public class MyEntry : Entry {}
public class RoundedBoxView : BoxView {}
```

**Extend an Existing Renderer** **Figure 5** extends the renderer that renders the Entry view for the iOS platform. You'd put this class in the iOS platform project. This renderer sets the color and style of the underlying native text field.

You can do just about anything you want in your renderer because you're referencing native APIs. If you want to view the sample that contains this snippet, see "Xamarin.Forms Custom Renderer" at bit.ly/1xTljmR.

**Create a Renderer from Scratch** You can create a brand-new renderer that doesn't extend any other renderers. You'll do a bit more work, but it makes sense to create one if you want to do any of these things:

- Replace the renderer of a view.
- Add a new view type to your solution.
- Add a native control or native page to your solution.

For example, if you want to add a native UIView control to a page in the iOS version of your app, you could add a custom renderer to your iOS project, as shown in **Figure 6**.

The general pattern that appears in this renderer ensures you can use it in virtualized layouts such as a list view, which I'll discuss later.

If you want to view the sample that contains this snippet, see bit.ly/xf-customrenderer.

**Add Properties to a Custom View** You can add properties to a custom view, but just make sure you make them bindable so you can bind them to properties in a view model or to other types of data. Here's a bindable property in the RoundedBoxView custom view:

```
public class RoundedBoxView : BoxView
{
  public static readonly BindableProperty CornerRadiusProperty =
    BindableProperty.Create<RoundedBoxView, double>(p => p.CornerRadius, 0);

  public double CornerRadius
  {
    get { return (double)base.GetValue(CornerRadiusProperty);}
    set { base.SetValue(CornerRadiusProperty, value);}
  }
}
```

To connect new properties to your renderer, override the OnElementPropertyChanged method of the renderer and add code that runs when the property changes:

```
protected override void OnElementPropertyChanged(object sender,
  System.ComponentModel.PropertyChangedEventArgs e)
{
  base.OnElementPropertyChanged(sender, e);
  if (e.PropertyName ==
    RoundedBoxView.CornerRadiusProperty.PropertyName)
      childView.Layer.CornerRadius = (float)this.Element.CornerRadius;
}
```

To learn more about creating custom views and custom renderers, see "Customizing Controls for Each Platform" at bit.ly/11pSFhL.

## Pages, Layouts and Views: The Building Blocks of Xamarin.Forms

I've showcased a few elements, but there are many more. This would be a good time to visit them.

Xamarin.Forms apps contain pages, layouts and views. A page contains one or more layouts, and a layout contains one or more views. The term view is used in Xamarin to describe what you might be used to calling a control. In total, the Xamarin.Forms framework contains five page types, seven layout types and 24 view types. You can read more about them at xamarin.com/forms. I'll visit some important page types later, but first I'll take a moment to review some of the layouts you can use in your app. Xamarin.Forms contains four primary layouts:

- **StackLayout:** The StackLayout positions child elements in a single line that can be oriented vertically or horizontally. You can nest StackLayouts to create complex visual hierarchies. You can control how views are arranged in a StackLayout by using the VerticalOptions and HorizontalOptions properties of each child view.
- **Grid:** The Grid arranges views into rows and columns. This layout is similar to the one you get with WPF and Silverlight except you can add spacing between rows and columns. You do that by using the RowSpacing and ColumnSpacing properties of the Grid.
- **RelativeLayout:** The RelativeLayout positions views by using constraints relative to other views.
- **AbsoluteLayout:** The AbsoluteLayout lets you position child views in two ways: at an absolute position, or proportionally relative to the parent. This can be useful if you plan to create split and overlaid hierarchies. **Figure 7** shows an example.

Note that all layouts give you a property named Children. You can use that property to access additional members. For example,

Figure 3 **Binding Views to Class Properties**

```
public LogInPage()
{
  Entry userEntry = new Entry { Placeholder = "Username" };
  userEntry.SetBinding(Entry.TextProperty, "UsernameText");

  Entry passEntry =
    new Entry { Placeholder = "Password", IsPassword = true };
  passEntry.SetBinding(Entry.TextProperty, "PasswordText");

  Button submit = new Button { Text = "Submit" };
  Content = new StackLayout
    {
      Padding = 20,
      VerticalOptions = LayoutOptions.Center,
      Children = { userEntry, passEntry, submit }
    };

  BindingContext = new LoginViewModel();
}
```

Figure 4 **Adding Markup to a Forms XAML Page**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  x:Class="Jason3.LogInPage"
  xmlns:local="clr-namespace:XamarinForms;assembly=XamarinForms">
    <StackLayout VerticalOptions="Center">
      <StackLayout.BindingContext>
        <local:LoginViewModel />
      </StackLayout.BindingContext>
      <Entry Text="{Binding UsernameText}" Placeholder="Username" />
      <Entry Text="{Binding PasswordText}"
        Placeholder="Password" IsPassword="true" />
      <Button Text="Login" Command="{Binding LoginCommand}" />
    </StackLayout>
</ContentPage>
```

you can use the Children property of the Grid layout to add and remove rows and columns, as well as specify row and column spans.

## Show Data in a Scrolling List

You can show data in a scrolling list by using a list view form (named ListView). This view performs well because the renderers for each cell in the list are virtualized. Because each cell is virtualized, it's important that you properly handle the OnElementChanged event of any custom renderers that you create for cells or lists by using a pattern similar to the one shown earlier.

First, define a cell, as shown in **Figure 8**. All data templates for a ListView must use a Cell as the root element.

Next, define a data source and set the ItemTemplate property of the ListView to a new data template. The data template is based on the MyCell class created earlier:

```
var items = new[] {
  new { Name = "Flower", Description = "A lovely pot of flowers." },
  new { Name = "Tuna", Description = "A can of tuna!" },
  // ... Add more items
};
var listView = new ListView
{
  ItemsSource = items,
  ItemTemplate = new DataTemplate(typeof(MyCell))
};
```

You can do this in XAML by using the following markup:

```
<ListView ItemsSource="{Binding Items}">
  <ListView.ItemTemplate>
    <ViewCell>
      <StackLayout VerticalOptions="center">
      <Label Text="{Binding Name}" />
      <Label Text="{Binding Description}" />
      </StackLayout>
    </ViewCell>
  </ListView.ItemTemplate>
</ListView>
```

## Navigate Between Pages

Most apps contain more than one page, so you'll need to enable users to navigate from one page to another. The following pages have built-in support for page navigation and support full-screen modal page presentation:

- TabbedPage
- MasterDetailPage
- NavigationPage
- CarouselPage

You can add your pages as children to any of these four pages and get navigation for free.

**The Tabbed Page** A TabbedPage shows an array of tabs across the top of a screen. Assuming your PCL project contains pages named LogInPage, DirectoryPage and AboutPage, you could add them all to a TabbedPage by using the following code:

```
var tabbedPage = new TabbedPage
{
  Children =
  {
    new LoginPage { Title = "Login", Icon = "login.png" },
    new DirectoryPage { Title = "Directory", Icon = "directory.png" },
    new AboutPage { Title = "About", Icon = "about.png" }
  }
};
```

In this case, it's important to set the Title and Icon property of each page so that something appears on page tabs. Not all platforms render icons. That depends on the platform's tab design.

If you opened this page on a mobile device, the first tab would appear as selected. However, you can change that by setting the CurrentPage property of the TabbedPage page.

**The NavigationPage** A NavigationPage manages the navigation and UX of a stack of pages. This page offers you the most common type of mobile app navigation pattern. Here's how you would add your pages to it:

```
var loginPage = new LoginPage();

var navigationPage = new NavigationPage(loginPage);

loginPage.LoginSuccessful += async (o, e) => await
  navigationPage.PushAsync(new DirectoryPage());
```

Notice the use of PushAsync as a way to navigate users to a specific page (in this case the DirectoryPage). In a NavigationPage, you "push" pages onto a stack and then "pop" them off as users navigate backward to the previous page.

Figure 5 **Extending an Existing Renderer**

```
[assembly: ExportRenderer (typeof (MyEntry), typeof (MyEntryRenderer))]

namespace CustomRenderer.iOS
{
  public class MyEntryRenderer : EntryRenderer
  {
    protected override void OnElementChanged
      (ElementChangedEventArgs<Entry> e)
    {
      base.OnElementChanged (e);

      if (e.OldElement == null) {

        var nativeTextField = (UITextField)Control;
        nativeTextField.BackgroundColor = UIColor.Gray;
        nativeTextField.BorderStyle = UITextBorderStyle.Line;
      }
    }
  }
}
```

Figure 6 **Adding a Native UIView Control to an iOS App with a Custom Renderer**

```
[assembly: ExportRendererAttribute(typeof(RoundedBoxView),
  typeof(RoundedBoxViewRenderer))]

namespace RBVRenderer.iOS
{
  public class RoundedBoxViewRenderer :
    ViewRenderer<RoundedBoxView,UIView>
  {
    protected override void OnElementChanged(
      ElementChangedEventArgs<RoundedBoxView> e)
    {
      base.OnElementChanged(e);

      var rbvOld = e.OldElement;
      if (rbvOld != null)
      {
        // Unhook any events from e.OldElement here.
      }

      var rbv = e.NewElement;
      if (rbv != null)
      {
        var shadowView = new UIView();
        // Set properties on the UIView here.
        SetNativeControl(shadowView);
        // Hook up any events from e.NewElement here.
      }
    }
  }
}
```

Figure 7 **Using the AbsoluteLayout**

```
void AbsoluteLayoutView()
{
  var layout = new AbsoluteLayout();
  var leftHalfOfLayoutChild = new BoxView { Color = Color.Red };
  var centerAutomaticallySizedChild =
    new BoxView { Color = Color.Green };
  var absolutelyPositionedChild = new BoxView { Color = Color.Blue };
  layout.Children.Add(leftHalfOfLayoutChild,
    new Rectangle(0, 0, 0.5, 1),
    AbsoluteLayoutFlags.All);
  layout.Children.Add(centerAutomaticallySizedChild,
    new Rectangle(
    0.5, 0.5, AbsoluteLayout.AutoSize, AbsoluteLayout.AutoSize),
    AbsoluteLayoutFlags.PositionProportional);
  layout.Children.Add(
    absolutelyPositionedChild, new Rectangle(10, 20, 30, 40));
}
```

The PushAsync and PopAsync methods of a NavigationPage are asynchronous so your code should await them and not push or pop any new pages while the task is running. The task of each method returns after the animation of a push or pop completes.

> You create a more engaging experience by animating the views on a page, which can be done in two ways.

For convenience, all Xamarin.Forms views, layouts and pages contain a Navigation property. This property is a proxy interface that contains the PushAsync and PopAsync methods of a NavigationPage instance. You could use that property to navigate to a page instead of calling the PushAsync and PopAsync methods on the NavigationPage instance directly. You can also use the NavigationProperty to get to the PushModalAsync and PopModalAsync methods. This is useful if you want to replace the contents of the entire screen with a new modal page. You don't have to have a NavigationPage in the parent stack of a view to use the Navigation property of a view, but nonmodal PushAsync/PopAsync operations might fail.

**A Note about Design Patterns** As a general practice, consider adding NavigationPages as children to TabbedPages, and TabbedPages as children to MasterDetailPages. Certain types of patterns can cause an undesirable UX. For example, most platforms advise against adding a TabbedPage as a child of a NavigationPage.

Figure 8 **Defining a Cell for a ListView**

```
public class MyCell : ViewCell
{
  public MyCell()
  {
    var nameLabel = new Label();
    nameLabel.SetBinding(Label.TextProperty, "Name");
    var descLabel = new Label();
    descLabel.SetBinding(Label.TextProperty, "Description");

    View = new StackLayout
    {
      VerticalOptions = LayoutOptions.Center,
      Children = { nameLabel, descLabel }
    };
  }
}
```

## Animate Views in a Page

You create a more engaging experience by animating the views on a page, which can be done in two ways. Either choose built-in animations that come with Xamarin.Forms or build one yourself by using the animation API.

For example, you could create a fading effect by calling the FadeTo animation of a view. The FadeTo animation is built into a view so it's easy to use:

```
async Task SpinAndFadeView(View view)
{
  await view.FadeTo(20, length: 200, easing: Easing.CubicInOut);
}
```

You can chain a series of animation together by using the await keyword. Each animation executes after the previous one completes. For example, you could rotate a view just before you fade it into focus:

```
async Task SpinAndFadeView(View view)
{
  await view.RotateTo(180);
  await view.FadeTo(20, length: 200, easing: Easing.CubicInOut);
}
```

If you have trouble achieving the effect you want, you can use the full animation API. In the following code, the view fades halfway through the rotation:

```
void SpinAndFadeView(View view)
{
  var animation = new Animation();
  animation.Add(0, 1, new Animation(
    d => view.Rotation = d, 0, 180, Easing.CubicInOut));
  animation.Add(0.5, 1, new Animation(
    d => view.Opacity = d, 1, 0, Easing.Linear));
  animation.Commit(view, "FadeAndRotate", length: 250);
}
```

This example composes each animation into a single Animation instance and then runs the entire animation sequence by using the Commit method. Because this animation isn't tied to a specific view, you can apply the animation to any of them.

## Wrapping Up

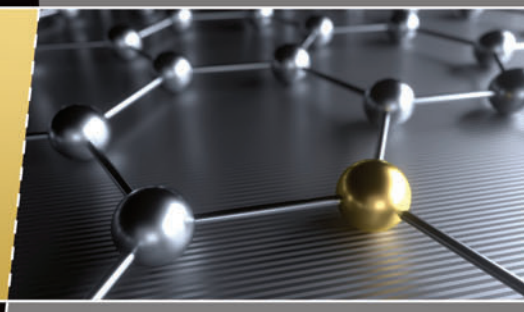
Xamarin.Forms is an exciting new way to build cross-platform native mobile apps. Use it to build a UI that renders natively across iOS, Android and Windows Phone. You can share almost all of your code between platforms.

Xamarin Inc. built Xamarin and Xamarin.Forms to make it possible for C# developers to jump into mobile development virtually overnight. If you've developed for the Windows Runtime, WPF or Silverlight, then you'll find that Xamarin.Forms is an easy bridge into the world of cross-platform native mobile development. You can install Xamarin today and immediately start to use C# to build beautiful native apps that run on iOS, Android and Windows Phone devices. ■

**Jason Smith** *is an engineering technical lead at Xamarin Inc. in San Francisco, currently leading the Xamarin.Forms project. He was one of the principal architects of Xamarin.Forms, prior to which he contributed to the Xamarin Studio project and was part of the initial research that lead to the creation of Xamarin Test Cloud.*

**Thanks** to the following Microsoft technical expert for reviewing this article: *Norm Estabrook*

# Switch to Amyuni PDF



## Create & Edit PDFs in .Net - ActiveX - WinRT

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

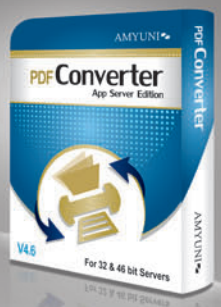

## Complete Suite of Accurate PDF Components

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


## Advanced HTML to PDF & XAML

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


## High Performance PDF Printer For Desktops and Servers




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

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

AMYUNI

**All development tools available at**

# www.amyuni.com


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

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

# Adding a Code Fix to Your Roslyn Analyzer

Alex Turner

If you followed the steps in my previous article, "Use Roslyn to Write a Live Code Analyzer for Your API" (msdn.microsoft.com/magazine/dn879356), you wrote an analyzer that displays live errors for invalid regular expression (regex) pattern strings. Each invalid pattern gets a red squiggle in the editor, just as you'd see for compiler errors, and the squiggles appear live as you type your code. This is made possible by the new .NET Compiler Platform ("Roslyn") APIs, which power the C# and Visual Basic editing experiences in Visual Studio 2015 Preview.

Can you do more? If you've got the domain knowledge to see not just what's wrong but also how to fix it, you can suggest the relevant code fix through the new Visual Studio light bulb. This code fix will allow a developer using your analyzer to not just find an error in his code—he can also clean it up instantly.

In this article, I'll show you how to add a code fix provider to your regex diagnostic analyzer that offers fixes at each regex squiggle. The fix will be added as an item in the light bulb menu, letting the user preview the fix and apply it to her code automatically.

## Picking Up Where You Left Off

To get started, be sure you've followed the steps in the previous article. In that article, I showed you how to write the first half of your analyzer, which generates the diagnostic squiggles under each invalid regex pattern string. That article walked you through:

- Installing Visual Studio 2015 Preview, its SDK and the relevant Roslyn VSIX packages.
- Creating a new Diagnostic with Code Fix project.
- Adding code to DiagnosticAnalyzer.cs to implement the invalid regex pattern detection.

If you're looking to quickly catch up, check out **Figure 1**, which lists the final code for DiagnosticAnalyzer.cs.

This article discusses prerelease versions of Visual Studio 2015 and the .NET Compiler Platform ("Roslyn") SDK. All related information is subject to change.

**This article discusses:**

- Transforming immutable syntax trees
- Getting the diagnostic ID
- Producing a code fix for a given diagnostic
- Updating the code

**Technologies discussed:**

.NET Compiler Platform, Visual Studio 2015 Preview

## Transforming Immutable Syntax Trees

Last time, when you wrote the diagnostic analyzer to detect invalid regex patterns, the first step was to use the Syntax Visualizer to identify patterns in the syntax trees that indicated problem code. You then wrote an analysis method that ran each time the relevant node type was found. The method checked for the pattern of syntax nodes that warranted an error squiggle.

Writing a fix is a similar process. You deal in syntax trees, focusing on the desired new state of the code files after the user applies your fix. Most code fixes involve adding, removing or replacing syntax nodes from the current trees to produce new syntax trees. You can operate directly on syntax nodes or use APIs that let you make project-wide changes, such as renames.

One very important property to understand about the syntax nodes, trees and symbols in the .NET Compiler Platform is that they're immutable. Once a syntax node or tree is created, it can't be modified—a given tree or node object will always represent the same C# or Visual Basic code.

Immutability in an API for transforming source code may seem counterintuitive. How can you add, remove and replace the child nodes in a syntax tree if neither the tree nor its nodes can be changed? It's helpful here to consider the .NET String type, another immutable type you use most likely every day. You perform operations to transform strings quite often, concatenating them together and even replacing substrings using String.Replace. However, none of these operations actually change the original string object. Instead, each call returns a new string object that represents the new state of the string. You can assign this new object back to your original variable, but any method you passed the old string to will still have the original value.

Figure 1 **The Complete Code for DiagnosticAnalyzer.cs**

```
using System;
using System.Collections.Immutable;
using Microsoft.CodeAnalysis;
using Microsoft.CodeAnalysis.CSharp;
using Microsoft.CodeAnalysis.CSharp.Syntax;
using Microsoft.CodeAnalysis.Diagnostics;

namespace RegexAnalyzer
{
  [DiagnosticAnalyzer(LanguageNames.CSharp)]
  public class RegexAnalyzerAnalyzer : DiagnosticAnalyzer
  {
    public const string DiagnosticId = "Regex";
    internal const string Title = "Regex error parsing string argument";
    internal const string MessageFormat = "Regex error {0}";
    internal const string Category = "Syntax";

    internal static DiagnosticDescriptor Rule =
      new DiagnosticDescriptor(DiagnosticId, Title, MessageFormat,
      Category, DiagnosticSeverity.Error, isEnabledByDefault: true);

    public override ImmutableArray<DiagnosticDescriptor>
      SupportedDiagnostics { get { return ImmutableArray.Create(Rule); } }

    public override void Initialize(AnalysisContext context)
    {
      context.RegisterSyntaxNodeAction(
        AnalyzeNode, SyntaxKind.InvocationExpression);
    }

    private void AnalyzeNode(SyntaxNodeAnalysisContext context)
    {
      var invocationExpr = (InvocationExpressionSyntax)context.Node;

      var memberAccessExpr =
        invocationExpr.Expression as MemberAccessExpressionSyntax;
      if (memberAccessExpr?.Name.ToString() != "Match") return;

      var memberSymbol = context.SemanticModel.
        GetSymbolInfo(memberAccessExpr).Symbol as IMethodSymbol;
      if (!memberSymbol?.ToString().StartsWith(
        "System.Text.RegularExpressions.Regex.Match") ?? true) return;

      var argumentList = invocationExpr.ArgumentList as ArgumentListSyntax;
      if ((argumentList?.Arguments.Count ?? 0) < 2) return;

      var regexLiteral =
        argumentList.Arguments[1].Expression as LiteralExpressionSyntax;
      if (regexLiteral == null) return;

      var regexOpt = context.SemanticModel.GetConstantValue(regexLiteral);
      if (!regexOpt.HasValue) return;

      var regex = regexOpt.Value as string;
      if (regex == null) return;

      try
      {
        System.Text.RegularExpressions.Regex.Match("", regex);
      }
      catch (ArgumentException e)
      {
        var diagnostic =
          Diagnostic.Create(Rule, regexLiteral.GetLocation(), e.Message);
        context.ReportDiagnostic(diagnostic);
      }
    }
  }
}
```

**Adding a Parameter Node to an Immutable Tree** To explore how immutability applies to syntax trees, you'll perform a simple transform manually in the code editor, and see how it affects the syntax tree.

Inside Visual Studio 2015 Preview (with the Syntax Visualizer extension installed, see previous article), create a new C# code file. Replace all of its contents with the following code:

```
class C
{
  void M()

  }
}
```

Open up the Syntax Visualizer by choosing View | Other Windows | Roslyn Syntax Visualizer and click anywhere within the code file to populate the tree. In the Syntax Visualizer window, right-click the root CompilationUnit node and choose View Directed Syntax Graph. Visualizing this syntax tree results in a graph like the one in **Figure 2** (the graph shown here omits the gray and white trivia nodes that represent whitespace). The blue ParameterList syntax node has two green child tokens representing its parentheses and no blue child syntax nodes, as the list contains no parameters.

The transform you'll simulate here is one that would add a new parameter of type int. Type the code "int i" within the parentheses of method M's parameter list and watch the changes within the Syntax Visualizer as you type:

```
class C
{
  void M(int i)
  {
  }
}
```

Note that even before you finish typing, when your incomplete code contains compile errors (shown in the Syntax Visualizer as nodes with a red background), the tree is still coherent, and the compiler guesses that your new code will form a valid Parameter node. This resilience of the syntax trees to compiler errors is what allows IDE features and your diagnostics to work well against incomplete code.

Right-click on the root CompilationUnit node again and generate a new graph, which should look like **Figure 3** (again, depicted here without trivia).

Note that the ParameterList now has three children, the two parenthesis tokens it had before, plus a new Parameter syntax node. As you typed "int i" in the editor, Visual Studio replaced the document's previous syntax tree with this new syntax tree that represents your new source code.

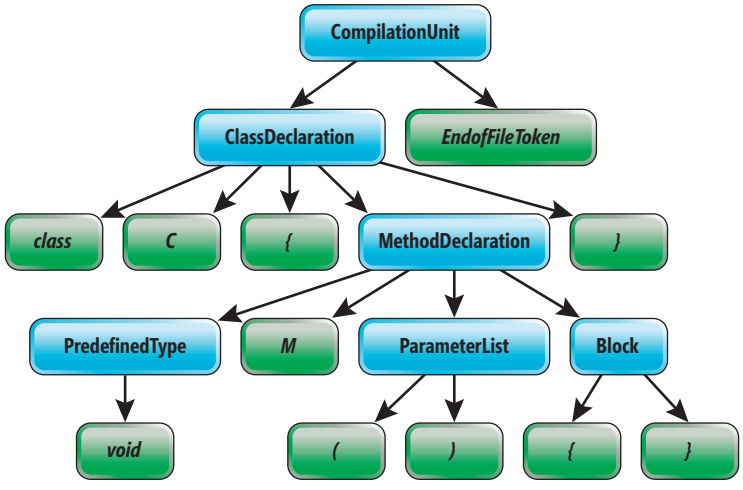



Figure 2 **Syntax Tree Before the Transform**

Performing a full replacement works well enough for small strings, which are single objects, but what about syntax trees? A large code file may contain thousands or tens of thousands of syntax nodes, and you certainly don't want all of those nodes to be recreated every time someone types a character within a file. That would generate tons of orphaned objects for the garbage collector to clean up and seriously hurt performance.

Luckily, the immutable nature of the syntax nodes also provides the escape here. Because most of the nodes in the document aren't affected when you make a small change, those nodes can be safely reused as children in the new tree. The behind-the-scenes internal node that stores the data for a given syntax node points only downward to the node's children. Because those internal nodes don't have parent pointers, it's safe for the same internal node to show up over and over again in many iterations of a given syntax tree, as long as that part of the code remains the same.

This node reuse means that the only nodes in a tree that need to be recreated on each keystroke are those with at least one descendant that has changed, namely the narrow chain of ancestor nodes up to the root, as depicted in **Figure 4**. All other nodes are reused as is.

In this case, the core change is to create your new Parameter node and then replace the ParameterList with a new ParameterList that has the new Parameter inserted as a child node. Replacing the ParameterList also requires replacing the chain of ancestor nodes, as each ancestor's list of child nodes changes to include the replaced node. Later in this article, you'll do that kind of replacement for your regex analyzer with the SyntaxNode.ReplaceNode method, which takes care of replacing all the ancestor nodes for you.

You've now seen the general pattern for planning a code fix: You start with code in the before state that triggers the diagnostic. Then you manually make the changes the fix should make, observing the effect on the syntax tree. Finally, you work out the code needed to create the replacement nodes and return a new syntax tree that contains them.

Be sure you've got your project open, containing the diagnostic you created last time. To implement your code fix, you'll dig into CodeFixProvider.cs.

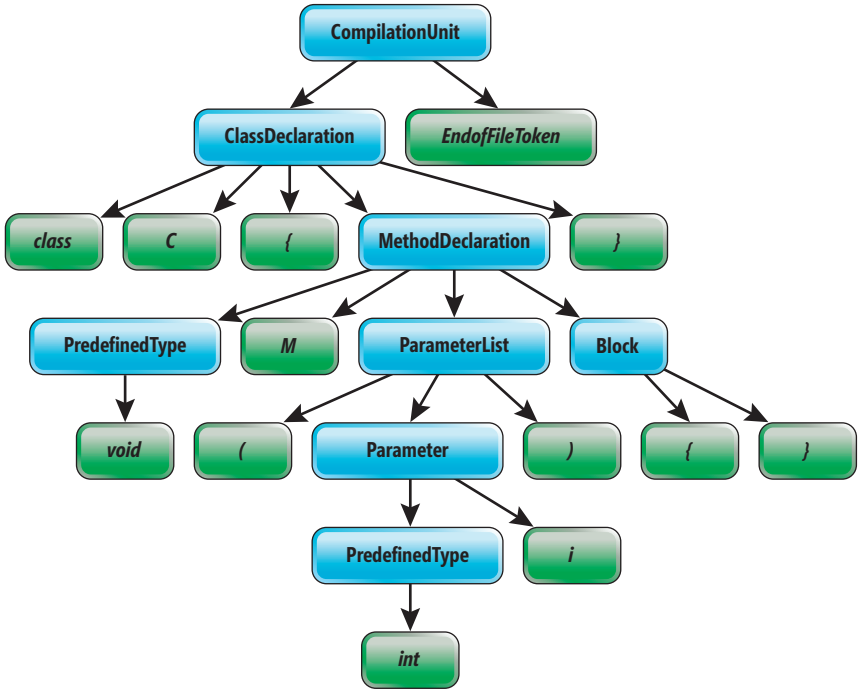



Figure 3 **Syntax Tree After the Transform**

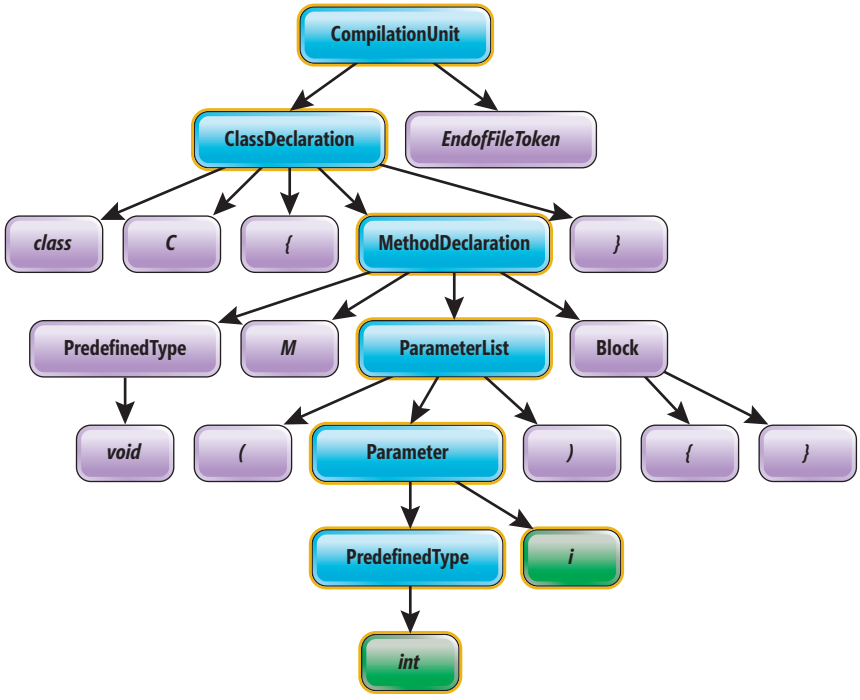



Figure 4 **The Ancestor Nodes Replaced During the Transform**

## GetFixableDiagnosticIds Method

Fixes and the diagnostics they resolve are loosely coupled by diagnostic IDs. Each code fix targets one or more diagnostic IDs. Whenever Visual Studio sees a diagnostic with a matching ID, it will ask your code fix provider if you have code fixes to offer. Loose coupling based on the ID string allows one analyzer to provide a fix for a diagnostic produced by someone else's analyzer, or even a fix for built-in compiler errors and warnings.

In this case, your analyzer produces both the diagnostic and the code fix. You can see that the GetFixableDiagnosticIds method is already returning the diagnostic ID you defined in your Diagnostic type, so there's nothing to change here.

## ComputeFixesAsync Method

The ComputeFixesAsync method is the main driver for the code fix. This method is called whenever one or more matching Diagnostics are found for a given span of code.

You can see that the template's default implementation of the ComputeFixesAsync method pulls out the first diagnostic from the context (in most cases, you only expect one), and gets the diagnostic's span. The next line then searches up the syntax tree from that span to find the nearest type declaration node. In the case of the default template's rule, that's the relevant node whose contents needed fixing.

In your case, the diagnostic analyzer you wrote was looking for invocations to see if they were calls to Regex.Match. To help share logic between your diagnostic and your code fix, change the type mentioned in the tree search's OfType filter to find that same InvocationExpressionSyntax node. Rename the local variable to "invocationExpr," as well:

```
var invocationExpr = root.FindToken(
  diagnosticSpan.Start).Parent.AncestorsAndSelf()
  .OfType<InvocationExpressionSyntax>().First();
```

You now have a reference to the same invocation node with which the diagnostic analyzer started. In the next statement, you pass this node to the method that will calculate the code changes you'll be suggesting for this fix. Rename that method from MakeUppercaseAsync to FixRegexAsync and change the fix description to Fix regex:

```
context.RegisterFix(
  CodeAction.Create("Fix regex", c => FixRegexAsync(
  context.Document, invocationExpr, c)), diagnostic);
```

Each call to the context's RegisterFix method associates a new code action with the diagnostic squiggle in question, and will produce a menu item inside the light bulb. Note that you're not actually calling the FixRegexAsync method that performs the code transform yet. Instead, the method call is wrapped in a lambda expression that Visual Studio can call later. This is because the result of your transform is only needed when the user actually selects your Fix regex item. When the fix item is highlighted or chosen, Visual Studio invokes your action to generate a preview or apply the fix. Until then, Visual Studio avoids running your fix method, just in case you're performing expensive operations, such as solution-wide renames.

Note that a code fix provider isn't obligated to produce a code fix for every instance of a given diagnostic. It's often the case that you have a fix to suggest only for some of the cases your analyzer can squiggle. If you'll only have fixes some of the time, you should first test in ComputeFixesAsync any conditions you need to determine whether you can fix the specific situation. If those conditions aren't met, you should return from ComputeFixesAsync without calling RegisterFix.

For this example, you'll offer a fix for all instances of the diagnostic, so there are no more conditions to check.

## FixRegexAsync Method

Now you get to the heart of the code fix. The FixRegexAsync method as currently written takes a Document and produces an updated

Solution. While diagnostic analyzers look at specific nodes and symbols, code fixes can change code across the entire solution. You can see that the template code here is calling Renamer.RenameSymbolAsync, which changes not just the symbol's type declaration, but also any references to that symbol throughout the solution.

In this case, you only intend to make local changes to the pattern string in the current document, so you can change the method's return type from Task<Solution> to Task<Document>. This signature is still compatible with the lambda expression in ComputeFixesAsync, as CodeAction.Create has another overload that accepts a Document rather than a Solution. You'll also need to update the typeDecl parameter to match the InvocationExpressionSyntax node you're passing in from the ComputeFixesAsync method:

```
private async Task<Document> FixRegexAsync(Document document,
  InvocationExpressionSyntax invocationExpr, CancellationToken cancellationToken)
```

Because you don't need any of the "make uppercase" logic, delete the body of the method, as well.

**Finding the Node to Replace** The first half of your fixer will look much like the first half of your diagnostic analyzer—you need to dig into the InvocationExpression to find the relevant parts of the method call that will inform your fix. In fact, you can just copy in the first half of the AnalyzeNode method down to the try-catch block. Skip the first line, though, as you already take invocationExpr as a parameter. Because you know this is code for which you've successfully found a diagnostic, you can remove all of the "if" checks. The only other change to make is to fetch the semantic model from the Document argument, as you no longer have a context that provides the semantic model directly.

When you finish those changes, the body of your FixRegexAsync method should look like this:

```
var semanticModel = await document.GetSemanticModelAsync(cancellationToken);

var memberAccessExpr =
  invocationExpr.Expression as MemberAccessExpressionSyntax;
var memberSymbol =
  semanticModel.GetSymbolInfo(memberAccessExpr).Symbol as IMethodSymbol;
var argumentList = invocationExpr.ArgumentList as ArgumentListSyntax;
var regexLiteral =
  argumentList.Arguments[1].Expression as LiteralExpressionSyntax;
var regexOpt = semanticModel.GetConstantValue(regexLiteral);
var regex = regexOpt.Value as string;
```

**Generating the Replacement Node** Now that you again have regexLiteral, which represents your old string literal, you need to generate the new one. Calculating exactly what string you need to fix an arbitrary regex pattern is a large task that's far beyond the scope of this article. As a stand-in for now, you'll just use the string valid regex, which is indeed a valid regex pattern string. If you decide to go further on your own, you should start small and target your fix at very particular regex problems.

The low-level way to produce new syntax nodes to substitute into your tree is through the members on SyntaxFactory. These methods let you create every type of syntax node in exactly the shape you choose. However, often it proves easier to just parse the expression you want from text, letting the compiler do all the heavy lifting to create the nodes. To parse a snippet of code, just call SyntaxFactory.ParseExpression and specify the code for a string literal:

```
var newLiteral = SyntaxFactory.ParseExpression("\"valid regex\"");
```

Figure 5 **The Complete Code for CodeFixProvider.cs**

```
using System.Collections.Immutable;
using System.Composition;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;
using Microsoft.CodeAnalysis;
using Microsoft.CodeAnalysis.CodeFixes;
using Microsoft.CodeAnalysis.CodeActions;
using Microsoft.CodeAnalysis.CSharp;
using Microsoft.CodeAnalysis.CSharp.Syntax;
using Microsoft.CodeAnalysis.Formatting;

namespace RegexAnalyzer
{
  [ExportCodeFixProvider("RegexAnalyzerCodeFixProvider",
    LanguageNames.CSharp), Shared]
  public class RegexAnalyzerCodeFixProvider : CodeFixProvider
  {
    public sealed override ImmutableArray<string> GetFixableDiagnosticIds()
    {
      return ImmutableArray.Create(RegexAnalyzer.DiagnosticId);
    }

    public sealed override FixAllProvider GetFixAllProvider()
    {
      return WellKnownFixAllProviders.BatchFixer;
    }

    public sealed override async Task ComputeFixesAsync(CodeFixContext context)
    {
      var root =
        await context.Document.GetSyntaxRootAsync(context.CancellationToken)
        .ConfigureAwait(false);

      var diagnostic = context.Diagnostics.First();
      var diagnosticSpan = diagnostic.Location.SourceSpan;

      // Find the invocation expression identified by the diagnostic.
      var invocationExpr =
        root.FindToken(diagnosticSpan.Start).Parent.AncestorsAndSelf()
        .OfType<InvocationExpressionSyntax>().First();

      // Register a code action that will invoke the fix.
      context.RegisterFix(
        CodeAction.Create("Fix regex", c =>
        FixRegexAsync(context.Document, invocationExpr, c)), diagnostic);
    }

    private async Task<Document> FixRegexAsync(Document document,
      InvocationExpressionSyntax invocationExpr,
      CancellationToken cancellationToken)
    {
      var semanticModel =
        await document.GetSemanticModelAsync(cancellationToken);

      var memberAccessExpr =
        invocationExpr.Expression as MemberAccessExpressionSyntax;
      var memberSymbol =
        semanticModel.GetSymbolInfo(memberAccessExpr).Symbol as IMethodSymbol;
      var argumentList = invocationExpr.ArgumentList as ArgumentListSyntax;
      var regexLiteral =
        argumentList.Arguments[1].Expression as LiteralExpressionSyntax;
      var regexOpt = semanticModel.GetConstantValue(regexLiteral);
      var regex = regexOpt.Value as string;

      var newLiteral = SyntaxFactory.ParseExpression("\"valid regex\"")
        .WithLeadingTrivia(regexLiteral.GetLeadingTrivia())
        .WithTrailingTrivia(regexLiteral.GetTrailingTrivia())
        .WithAdditionalAnnotations(Formatter.Annotation);
      var root = await document.GetSyntaxRootAsync();
      var newRoot = root.ReplaceNode(regexLiteral, newLiteral);

      var newDocument = document.WithSyntaxRoot(newRoot);

      return newDocument;
    }
  }
}
```

This new literal would work well as a replacement in most cases, but it's missing something. If you recall, syntax tokens can have attached trivia that represents whitespace or comments. You'll need to copy over any trivia from the old literal expression to ensure you don't delete any spacing or comments from the old code. It's also good practice to tag new nodes you create with the "Formatter" annotation, which informs the code fix engine that you want your new node formatted according to the end user's code style settings. You'll need to add a using directive for the Microsoft.CodeAnalysis.Formatting namespace. With these additions, your ParseExpression call looks like this:

```
var newLiteral = SyntaxFactory.ParseExpression("\"valid regex\"")
  .WithLeadingTrivia(regexLiteral.GetLeadingTrivia())
  .WithTrailingTrivia(regexLiteral.GetTrailingTrivia())
  .WithAdditionalAnnotations(Formatter.Annotation);
```

**Swapping the New Node into the Syntax Tree** Now that you have a new syntax node for the string literal, you can replace the old node within the syntax tree, producing a new tree with a fixed regex pattern string.

First, you get the root node from the current document's syntax tree:

```
var root = await document.GetSyntaxRootAsync();
```

Now, you can call the ReplaceNode method on that syntax root to swap out the old syntax node and swap in the new one:

```
var newRoot = root.ReplaceNode(regexLiteral, newLiteral);
```

Remember that you're generating a new root node here. Replacing any syntax node also requires you to replace its parents all the way up to the root. As you saw before, all syntax nodes in the .NET Compiler Platform are immutable. This replacement operation actually just returns a new root syntax node with the target node and its ancestors replaced as directed.

Now that you have a new syntax root with a fixed string literal, you can walk up one more level of the tree to produce a new Document object that contains your updated root. To replace the root, use the WithSyntaxRoot method on the Document:

```
var newDocument = document.WithSyntaxRoot(newRoot);
```

This is the same With API pattern you just saw when calling WithLeadingTrivia and other methods on the expression you parsed. You'll see this With pattern often when transforming existing objects in the Roslyn immutable object model. The idea is similar to the .NET String.Replace method that returns a new string object.

With the transformed document in hand, you can now return it from FixRegexAsync:

```
return newDocument;
```

Your code in CodeFixProvider.cs should now look like **Figure 5**.

**Trying It Out** That's it! You've now defined a code fix whose transform runs when users encounter your diagnostic and choose the fix from the light bulb menu. To try out the code fix, press F5 again in the main instance of Visual Studio and open up the console application. This time, when you place the cursor on your squiggle, you should see a light bulb appear to the left. Clicking on the light bulb should bring up a menu that contains the Fix regex code action you defined, as shown in **Figure 6**. This menu shows a preview with an inline diff between the old Document and the new Document you created, which represents the state of your code if you choose to apply the fix.

If you select that menu item, Visual Studio takes the new Document and adopts it as the current state of the editor buffer for that source file. Your fix has now been applied!

## Congratulations

You've done it! In about 70 total lines of new code, you identified an issue in your user's code, live, as he's typing, squiggled it red as an error, and surfaced a code fix that can clean it up. You transformed syntax trees and generated new syntax nodes along the way, all while operating within your familiar target domain of regular expressions.

While you can continuously refine the diagnostics and code fixes you write, I've found that analyzers built with the .NET Compiler Platform let you get quite a lot done within a short amount of time. Once you get comfortable building analyzers, you'll start to spot all sorts of common problems in your daily coding life and detect repetitive fixes you can automate.

What will you analyze? ■

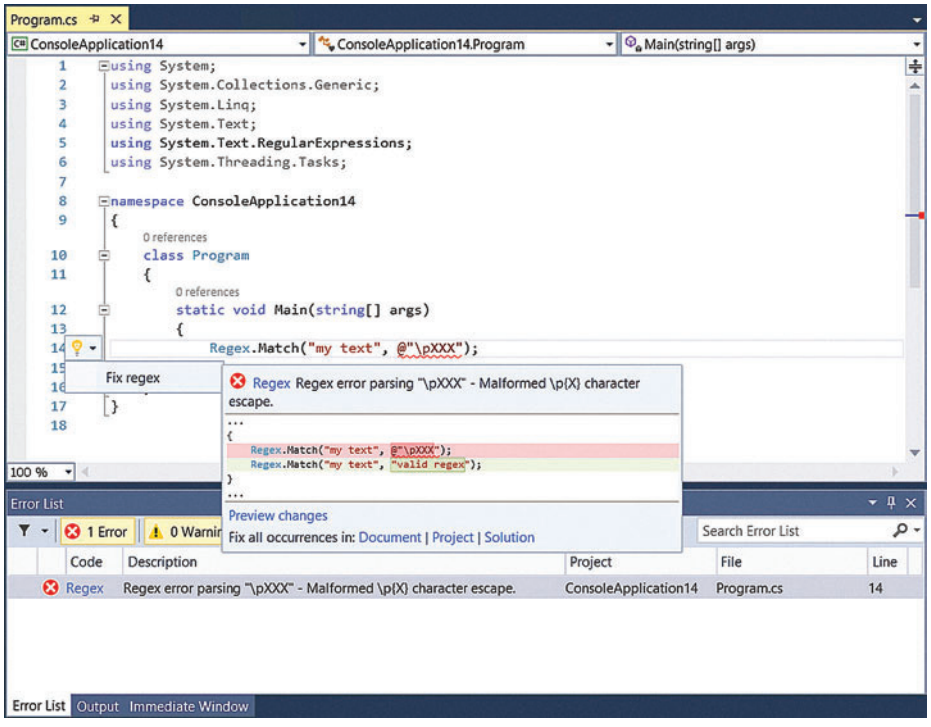



Figure 6 **Trying Out Your Code Fix**

**Alex Turner** *is a senior program manager for the Managed Languages team at Microsoft, where he's been brewing up C# and Visual Basic goodness on the .NET Compiler Platform ("Roslyn") project. He graduated with a Master of Science in Computer Science from Stony Brook University and has spoken at Build, PDC, TechEd, TechDays and MIX.*

**Thanks** to the following Microsoft technical experts for reviewing this article: *Bill Chiles and Lucian Wischik*

# The Rise of Event Stream-Oriented Systems

Christopher Bennage

It's all about data these days. Data helps us make informed decisions. Big Data helps us make informed and insightful decisions. Big streams of data help us make informed, insightful and timely decisions. These continuously flowing streams of data are often called event streams. It's increasingly common to build software systems whose primary purpose is to process event streams.

Even across different industries and domains, there's a discernable common architectural pattern around these event stream-oriented systems. This pattern for modern event stream-oriented systems plays the same fundamental role that the classic *n*-tier architecture held for traditional on-premises enterprise systems. I'll start off by exploring a thumbnail sketch of this nascent pattern.

## Identify the Pattern

First, I should clarify what is meant by the term event. Here, it means merely a bit of data signifying that something happened in a system. Events tend to be small in size, in the byte or kilobyte range. You'll also hear terms like message, telemetry or even just data in place of event.

Next, there are event producers. These producers could be almost anything—connected cars, smart thermostats, game consoles, personal fitness devices or even a software system generating self-diagnostic events. It's important to recognize, though, that in most of these systems, you're dealing with numerous event producers.

Many systems anticipate numbers of event producers in the tens of thousands and ranging into tens of millions or more. This means these systems tend to have both high volume and high velocity. High volume means there's a lot of data overall and high velocity means the data is generated frequently.

There are also event consumers. Consumers are the real heart of these types of systems. They're responsible for analyzing, interpreting and responding to events. The number of consumers in a typical system might range from a couple to a couple dozen. Events aren't routed to specific consumers. Each consumer is looking at the same set of events. In the context of Microsoft Azure, consumers are most likely cloud services.

Consider this example. There's an event stream representing financial transactions. The event producers in this scenario are point-of-sale systems in retail stores. One consumer owns the responsibility to analyze the stream for fraudulent activity and raise alerts. Another consumer analyzes the same stream to make just-in-time supply chain optimizations. Finally, a third consumer is responsible for translating the events into long-term cold storage for later analytics.

This article discusses:

- The Azure Event Hubs architecture
- How to configure Azure Event Hubs
- Implementing an event processor

Technologies discussed:

Microsoft Azure, Azure Event Hubs

# Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

vslive.com/lasvegas

## Las Vegas MARCH 16 – 20

BALLY'S HOTEL & CASINO LAS VEGAS, NV

## TRACKS INCLUDE:

- Visual Studio / .NET
- JavaScript/HTML5
- ASP.NET
- Windows 8.1/WinRT
- Cross-Platform Mobile Development
- Database and Analytics
- Cloud Computing
- SharePoint/Office


## Code on the Strip

### Register by February 25 and Save $300!

USE PROMO CODE VSLFEBTI

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

**Bonus Content**
ModernApps LIVE!
Presented in Partnership with Magenic

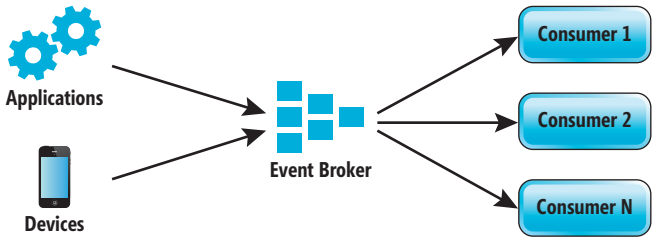



Figure 1 **The Azure Event Hub Architecture**

When combined with the reality of high-volume and high-velocity events, this pattern of producers and consumers presents a few interesting problems:

- How do you prevent event production surges from overwhelming consumers? That is, how should the system respond when the rate of event production starts to exceed the rate of consumption?
- Because event velocity is high, how can you scale an individual event consumer?

The key to the problem is to use an event broker (see **Figure 1**). This is precisely the role performed by the recently released Azure Event Hubs.

So how, exactly, does using a broker such as Event Hubs solve the problems I've outlined so far?

## Understand Event Hubs

Event Hubs provides the elasticity needed to absorb and persist events until downstream consumers can catch up. Event Hubs can effectively level out variability in the event stream rate so consumers don't have to worry about it. Without this leveling, a receiving consumer might become overwhelmed and begin to fail.

Using a broker isolates the event producers and event consumers from each other. This isolation is especially important in more sophisticated versions of the architectural pattern where additional intermediaries are necessary between the producers and the consumers. Event Hubs is a point of composition, a seam or boundary in the architecture. All components that interact through an Event Hub don't require specific knowledge of each other.

At this point, it might be easy to confuse Event Hubs with traditional enterprise messaging services that offer the same type of isolation. However, Event Hubs is different in several key ways that make it ideal for this architectural pattern.

## Independent Consumers

Event Hubs uses a publish and subscribe model; however, each consumer has an independent view of the same event stream. In some traditional messaging systems with multiple consumers, messages are copied for each interested consumer. This can be inefficient in terms of speed and space, but the benefit is that each consumer has its own "inbox." As a consumer processes messages, it removes them from its inbox. There's no affect on other consumers because they have their own copies in their own inboxes.

With Event Hubs, there's one set of immutable events and, because they're immutable, there only needs to be one copy of each event. Likewise, consumers never remove events from the system. All consumers are looking at the same set of events. Because of this, consumers own the responsibility of keeping track of where they are in the event stream. They do this by tracking their offset in the event stream. There's actually an API for this built into the SDK.

## Time-Based Retention

In traditional messaging systems, the consumer is responsible for telling the system when it's done with the message. The system can then get rid of the message. Because an Event Hubs consumer is responsible for tracking his own position within the event stream, how does an Event Hub know when the consumer is done with the events? In short, it doesn't. With Event Hubs, you configure a retention period and events are stored for that amount of time. This means events expire on their own, independent of any consumer action.

The implication of time-based retention is the consumer needs to examine and process events before they expire. With time-based retention, each consumer has pressure to keep up. Fortunately, the underlying design of Event Hubs lets individual consumers scale as necessary.

Event Hubs supports this by is physically partitioning the event stream. You set the number of partitions when provisioning an Event Hub. See the official documentation at bit.ly/11QAxOY for more details.

> Event Hubs provides the elasticity needed to absorb and persist events until downstream consumers can catch up.

As events are published to an Event Hub, they're placed in partitions. A given event resides in only one partition. Events are evenly distributed by default across partitions in a round-robin fashion. There are mechanisms for providing partition affinity. The most common lets you set a partition key property on an event, and all events with the same key will be delivered to the same partition.

How does a partitioned event stream help consumers with time-based retention? In the context of Event Hubs, the correct term is actually consumer group. The reason for calling it a group is each consumer really consists of multiple instances. Each group has one instance per partition. From this point, consumer group refers to the consumer as a whole and consumer instance refers to the member of the group interested in a particular partition.

This means a consumer group can process stream events in parallel. Each consumer instance in the group can process a partition independent of other instances. These consumer instances can all reside in one machine, with each consumer instance running in isolation from one another. They could all be distributed across multiple machines, even to the point of each consumer instance running on a dedicated box. This way, Event Hubs circumvents some of the typical problems associated with the classic pattern of competing consumers.

Isolation is a key concept here. First, you're isolating event producers and event consumers from each other, thus enabling flexible architecture composition, as well as load leveling. Second, consumer groups are isolated from each other, reducing the opportunity for cascading failures across consumer groups. Third, consumer instances in a given consumer group are isolated from each other to enable horizontal scaling for individual consumer groups.

## Use Event Hubs

There are several good tutorials for getting started with Event Hubs. Check out the official documentation at bit.ly/11QAxOY and follow the tutorial that uses the platform of your choice.

> Event Hubs uses a publish and subscribe model. However, each consumer has an independent view of the same event stream.

You'll need to provision an Event Hub first. The process is straightforward. You can easily try it out with a trial Azure account. In the Azure Management Portal, navigate to the Service Bus section. You'll need to create a Service Bus namespace if you don't already have one. After that, you'll see a tab called Event Hubs that has instructions for creating an Event Hub (see **Figure 2**).

You also need to set up a shared access policy for the Event Hub before you can begin. These policies manage security for an Event Hub. In the portal, navigate to the Event Hub you just created and select the Configure tab.

Choose Manage for the permissions and give the policy a name such as "super" or "do-not-use-in-production." After that, switch back to the Dashboard tab and click the Connection Information button at the bottom. You'll want to make note of the connection string there, as well as the name you gave your Event Hub.

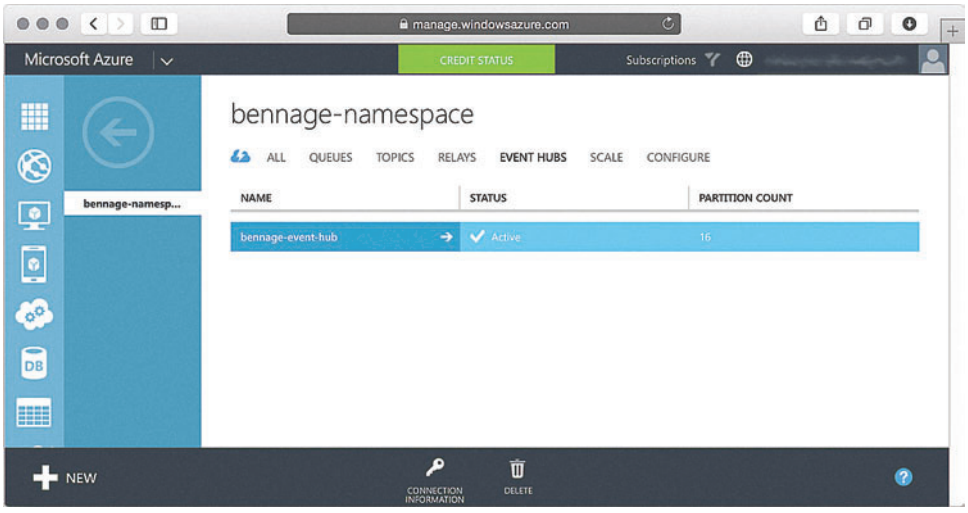



Figure 2 **Create an Event Hub**

## Produce Events

The code I'll show here uses the .NET SDK, but you can use any platform that supports HTTP or AMQP. You'll need to reference the Microsoft Azure Service Bus NuGet package. The classes you need are in the Microsoft.ServiceBus.Messaging namespace. All you need to do is create a client, create an event and send:

```
var client = EventHubClient.CreateFromConnectionString (
  connectionString,
  eventHubName);

var body = Encoding.UTF8.GetBytes("My first event");
var eventData = new EventData (body);

await client.SendAsync (eventData);
```

Despite the simplicity, there are a few interesting items to point out. The body of the event is just a byte array. Any consumer groups processing this event will need to know how to interpret those bytes. It's likely the consumer groups will need some sort of hint to determine how to deserialize the body. Before the event is sent, metadata can be attached:

```
eventData.Properties.Add ("event-type", "utf8string");
```

This means using keys and values that are well known by both producers and consumer groups. If you want to ensure a set of events is delivered to the same partition, you can set a partition key:

```
eventData.PartitionKey = "something-meaningful-to-your-domain";
```

You'll get better performance if events don't have affinity with partitions. In some cases, though, you'll want a set of related events routed to a single consumer instance for processing. Events in a given partition are guaranteed to be in the order they were received. Likewise, there's no easy way to guarantee the order of events across different partitions in an Event Hub. This is often the motivation for wanting events to have affinity to a particular partition.

For example, if you're enabling smart cars, you want all events for a given car to be in the same partition. You might choose the Vehicle Identification Number (VIN) for the partition key. Or your system might focus on smart buildings, with hundreds of devices in each building producing events. In that case, you might use the identity of the building itself as the partition key so all events from all devices in the same building land in the same partition.

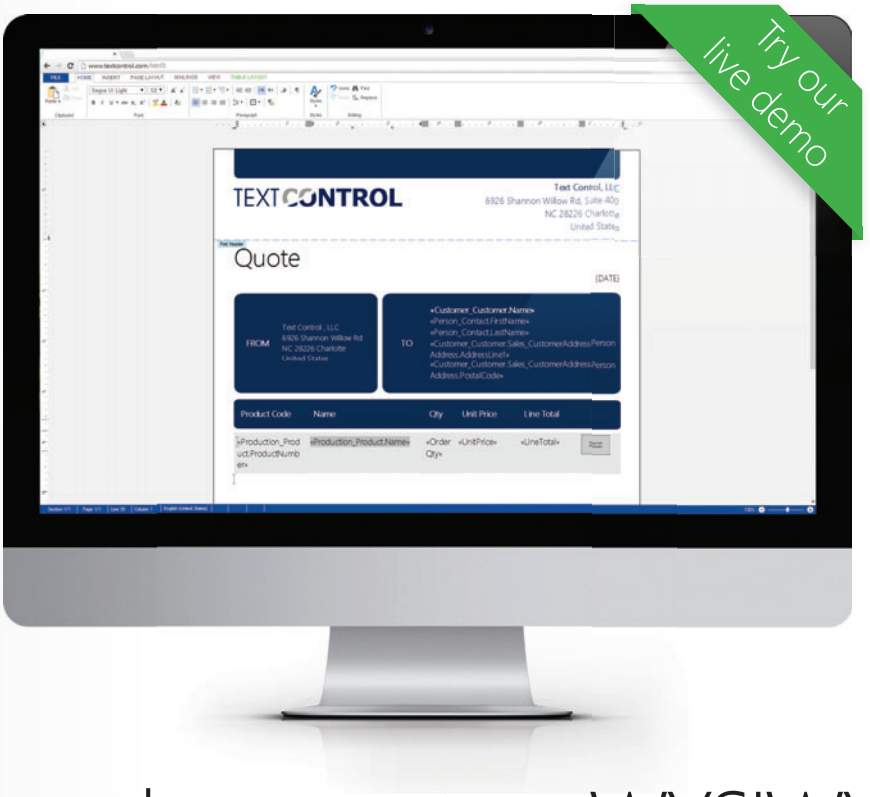
Overall, partition affinity is a dangerous practice and you should only use it carefully. A poor choice of partition key can result in an uneven event distribution across partitions. This could ultimately mean consumer groups would have trouble scaling. The good news is that many times you can change the system design to avoid the need for partition affinity.

## Consume Events

You may be concerned about how you'll manage all this. Your consumer groups need to keep track of their offset in the event stream. Each group needs to have an instance for each partition. Fortunately, there's an API for that.

Reference the NuGet package Microsoft Azure Service Bus

# Finally!




## Cross-browser, true WYSIWYG document editing for ASP.NET

- Edit and create MS Word documents
- Create and modify Adobe® PDF documents
- Create reports and mail merge templates
- Integrate with Microsoft® Visual Studio

Try it online and download your 30-day trial version today:
www.textcontrol.com/html5

Visual Studio Partner

HTML 5

TEXT CONTROL

Event Hub-EventProcessorHost. The classes you need are in the Microsoft.ServiceBus.Messaging namespace. Getting started is as simple as implementing a single interface: IEventProcessor.

Once you've implemented your event processor, you'll create an instance of EventProcessorHost to register your event processor. The host will handle all the grunt work for you. When it starts up, it will examine your Event Hub to see how many partitions it has. It will then create one instance of your event processor for each available partition.

There are three methods you need to implement. The first two are OpenAsync and CloseAsync. The host calls OpenAsync when the event processor instance is first granted a partition lease. This means the event processor instance has exclusive access to the partition for the consumer group in question. Likewise, the host calls CloseAsync when its lease is lost or when it's shutting down. While you're getting started, you can use a very simple implementation:

```
public Task OpenAsync(PartitionContext context)
{
  return Task.FromResult(true);
}

public Task CloseAsync(PartitionContext context, CloseReason reason)
{
  return Task.FromResult(true);
}
```

Both of these methods receive a PartitionContext argument. The remaining method receives it, as well. You can examine this argument if you want to view details about the specific partition leased to the event processor. The final method is where you actually receive the events (see **Figure 3**).

As you can see, this is straightforward. You receive an enumerable set of events you can iterate over and do whatever work is needed. You also have this invocation of context.CheckpointAsync at the end of the method. This tells the host you've successfully processed this set of events and you'd like to record a checkpoint. The checkpoint is the offset of the last event in the batch.

That's how your consumer group can keep track of which events have been processed for each partition. After a host is started, it tries to acquire a lease for any available partition. When it starts processing for a partition, it will examine the checkpoint information for that partition. Only events more recent than the last checkpointed offset are sent to their respective processors.

The host also provides automatic load leveling across machines. For example, let's say you have an Event Hub with 16 partitions. This means there will be 16 instances of your event processor—one for each partition. If you're running the host on a single machine, it creates all 16 instances on the same machine. If you start another host on a second machine and it's part of the same consumer group, the two hosts will begin to level the distribution of event processor instances across the two machines. There will ultimately be eight event processor instances per machine. Likewise, if you take down the second machine, then the first host takes back over the orphaned partitions.

Figure 3 **The Final Method that Delivers the Events**

```
public async Task ProcessEventsAsync (PartitionContext context,
IEnumerable<EventData> messages)
{
  foreach (var message in messages)
  {
    var eventType = message.Properties["event-type"];
    var bytes = message.GetBytes();

    if (eventType.ToString() == "utf8string") {
      var body = System.Text.Encoding.UTF8.GetString (bytes);
      // Do something interesting with the body
    } else {
      // Record that you don't know what to do with this event
    }
  }

  await context.CheckpointAsync();
  // This is not production-ready code
}
```

Assume your implementation of IEventProcessor is MyEventProcessor. Then instantiating the host can be as simple as this:

```
var host = new EventProcessorHost(
  hostName,
  eventHubName,
  consumerGroupName,
  eventHubConnectionString,
  checkpointConnectionString);

await host.RegisterEventProcessorAsync<MyEventProcessor>();
```

The eventHubConnectionString and eventHubName are the same values used when sending events in the previous example. It's best to use connection strings with shared access policies that restrict usage to just what's needed.

The hostName identifies the instance of the EventProcessorHost. When running the host in a cluster (meaning multiple machines), it's recommended you provide a name that reflects the identity of the machine on which it's running.

The consumerGroupName argument identifies the logical consumer group this host represents. There's a default consumer group you can reference using the constant EventHubConsumerGroup.DefaultGroupName. Any other name requires you first provision the consumer group. Do this by creating an instance of Microsoft.ServiceBus.NamespaceManager and using methods such as CreateConsumerGroupAsync.

Finally, you need to provide a connection string to an Azure Storage account using checkpointConnectionString. This storage account is where the host tracks all state regarding partitions and event offsets. This state is stored in blobs in plain text you can readily examine.

There are other Azure services that are integrated with Event Hubs out-of-the-box. Azure Stream Analytics (currently in Preview) provides a declarative SQL-like syntax for transforming and analyzing event streams originating in Event Hubs. Likewise, Event Hubs offers a spout for the very popular Apache Storm, now available as a Preview on Azure through HDInsight.

## Wrapping Up

The architectural pattern outlined here is just the beginning. When implementing a real-world system, there are numerous other concerns you'll need to take into consideration. These concerns involve advanced security, provisioning and management of event producers, protocol translation, outbound communication, and more. Nevertheless, you're now equipped with the foundational concepts necessary to build a system using an event broker such as Event Hubs. ■

**Christopher Bennage** *is a member of the Microsoft patterns & practices team. He likes to make things with computers.*

**Thanks** to the following Microsoft technical experts for reviewing this article: *Mostafa Elhemali and Dan Rosanova*


# Extreme Performance Linear Scalability

## For .NET & Java Apps

(Microsoft Azure Supported)

Cache data, reduce expensive database trips, and scale your apps to extreme transaction processing (XTP) with NCache.

### In-Memory Distributed Cache

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

### ASP.NET Optimization in Web Farms

- ASP.NET Session State storage
- ASP.NET View State cache
- ASP.NET Output Cache provider

### Runtime Data Sharing

- Powerful event notifications for pub/sub data sharing
- Continuous Query events

NCache™

FREE Download

sales@alachisoft.com US: +1 (925) 236 3830 www.alachisoft.com

# Build Better Software with Smart Unit Tests

Pratap Lakshman

The world of software development is hurtling toward ever-shortening release cycles. The time when software development teams could strictly sequence the functions of specification, implementation and testing in a waterfall model is long past. Developing high-quality software is hard in such a hectic world, and calls for a reevaluation of existing development methodologies.

To reduce the number of bugs in a software product, all team members must agree on what the software system is supposed to do, and that's a key challenge. Specification, implementation and testing have typically happened in silos, with no common medium of communication. Different languages or artifacts used for each made it difficult for them to co-evolve as the software implementation activity progresses, and so, while a specification document ought to connect the work of all the team members, that's rarely the case in reality. The original specification and the actual implementation might diverge, and the only thing holding everything together eventually is the code, which ends up embodying the ultimate specification and the various design decisions made *en route*. Testing attempts to reconcile this divergence by resorting to testing just a few well-understood end-to-end scenarios.

This situation can be improved. A common medium to specify the intended behavior of the software system is needed, one that can be shared across design, implementation and testing, and that's easy to evolve. The specification must be directly related to the code, and the medium be codified as an exhaustive suite of tests. Tool-based techniques enabled by Smart Unit Tests can help fulfill this need.

This article discusses:

- Test case generation
- Bounded exploration
- Parameterized unit testing

Technologies discussed:

Visual Studio 2015 Preview

## Smart Unit Tests

Smart Unit Tests, a feature of Visual Studio 2015 Preview (see **Figure 1**), is an intelligent assistant for software development, helping dev teams find bugs early and reduce test maintenance costs. It's based on previous Microsoft Research work called "Pex." Its engine uses white-box code analyses and constraint solving to synthesize precise test input values to exercise all code paths in the code under test, persist these as a compact suite of traditional unit tests with high coverage, and automatically evolve the test suite as the code evolves.

Moreover, and this is strongly encouraged, correctness properties specified as assertions in code can be used to further guide test case generation.

By default, if you do nothing more than just run Smart Unit Tests on a piece of code, the generated test cases capture the observed behavior of the code under test for each of the synthesized input values. At this stage, except for test cases causing runtime errors, the remaining are deemed to be passing tests—after all, that's the observed behavior.

Additionally, if you write assertions specifying the correctness properties of the code under test, then Smart Unit Tests will come up with test input values that can cause the assertions to fail, as well, each such input value uncovering a bug in the code, and thereby a failing test case. Smart Unit Tests can't come up with such correctness properties by itself; you'd write them based on your domain knowledge.

## Test Case Generation

In general, program analysis techniques fall between the following two extremes:

- **Static analysis** techniques verify that a property holds true on all execution paths. Because the goal is program verification, these techniques are usually overly conservative and flag possible violations as errors, leading to false positives.
- **Dynamic analysis** techniques verify that a property holds true on some execution paths. Testing takes a dynamic analysis approach that aims at detecting bugs, but it usually can't prove the absence of errors. Thus, these techniques often fail to detect all errors.

It might not be possible to detect bugs precisely when applying only static analysis or employing a testing technique that's unaware of the structure of the code. For example, consider the following code:

```
int Complicated(int x, int y)
{
  if (x == Obfuscate(y))
    throw new RareException();
  return 0;
}

int Obfuscate(int y)
{
  return (100 + y) * 567 % 2347;
}
```

Static analysis techniques tend to be conservative, so the non-linear integer arithmetic present in Obfuscate causes most static analysis techniques to issue a warning about a potential error in Complicated. Also, random testing techniques have very little chance of finding a pair of x and y values that trigger the exception.

Smart Unit Tests implements an analysis technique that falls between these two extremes. Similar to static analysis techniques, it proves that a property holds for most feasible paths. Similar to dynamic analysis techniques, it reports only real errors and no false positives.

Test case generation involves the following:

- Dynamically discovering all the branches (explicit and implicit) in the code under test.
- Synthesizing precise test input values that exercise those branches.
- Recording the output from the code under test for the said inputs.
- Persisting these as a compact test suite with high coverage.

**Figure 2** shows how it works using runtime instrumentation and monitoring and here are the steps involved:

1. The code under test is first instrumented and callbacks are planted that will allow the testing engine to monitor execution. The code is then run with the simplest relevant concrete input value (based on the type of the parameter). This represents the initial test case.
2. The testing engine monitors execution, computes coverage for each test case, and tracks how the input value flows through the code. If all paths are covered, the process stops; all exceptional behaviors are considered as branches, just like explicit branches in the code. If all paths haven't been covered yet, the testing engine picks a test case that reaches a program point from which an uncovered branch leaves, and determines how the branching condition depends on the input value.
3. The engine constructs a constraint system representing the condition under which control reaches to that program point and would then continue along the previously uncovered branch. It then queries a constraint solver to synthesize a new concrete input value based on this constraint.
4. If the constraint solver can determine a concrete input value for the constraint, the code under test is run with the new concrete input value.
5. If coverage increases, a test case is emitted.

Steps 2 through 5 are repeated until all branches are covered, or until preconfigured exploration bounds are exceeded.

This process is termed an "exploration." Within an exploration, the code under test can be "run" several times. Some of those runs increase coverage, and only the runs that increase coverage emit test cases. Thus, all tests that are generated exercise feasible paths.

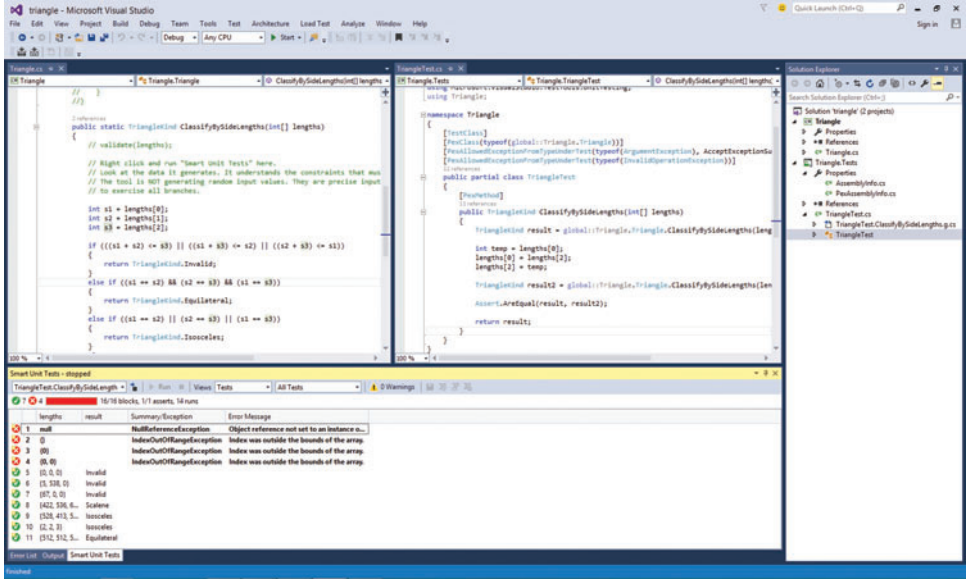



Figure 1 **Smart Unit Tests Is Fully Integrated into Visual Studio 2015 Preview**

## Bounded Exploration

If the code under test doesn't contain loops or unbounded recursion, exploration typically stops quickly because there are only a (small) finite number of execution paths to analyze. However, most interesting programs do contain loops or unbounded recursion. In such cases, the number of execution paths is (practically) infinite, and it's generally undecidable whether a statement is reachable. In other words, an exploration would take forever to analyze all execution paths of the program. Because test generation involves actually running the code under test, how do you protect from such runaway

exploration? That's where bounded exploration plays a key role. It ensures explorations stop after a reasonable amount of time. There are several tiered, configurable exploration bounds that are used:

- **Constraint solver bounds** limit the amount of time and memory the solver can use in searching for the next concrete input value.
- **Exploration path bounds** limit the complexity of the execution path being analyzed in terms of the number of branches taken, the number of conditions over the inputs that need to be checked, and the depth of the execution path in terms of stack frames.
- **Exploration bounds** limit the number of "runs" that don't yield a test case, the total number of runs permitted and an overall time limit after which exploration stops.

An important aspect to any tool-based testing approach being effective is rapid feedback, and all of these bounds have been preconfigured to enable rapid interactive use.

Furthermore, the testing engine uses heuristics to achieve high code coverage quickly by postponing solving hard constraint systems. You can let the engine quickly generate some tests for code on which you're working. However, to tackle the remaining hard test input generation problems, you can dial up the thresholds to let the testing engine crunch further on the complicated constraint systems.

## Parameterized Unit Testing

All program analysis techniques try to validate or disprove certain specified properties of a given program. There are different techniques for specifying program properties:

- **API Contracts** specify the behavior of individual API actions from the implementation's perspective. Their goal is to guarantee robustness, in the sense that operations don't crash and data invariants are preserved. A common problem of API contracts is their narrow view on individual API actions, which makes it difficult to describe system-wide protocols.
- **Unit Tests** embody exemplary usage scenarios from the perspective of a client of the API. Their goal is to guarantee functional correctness, in the sense that the interplay of several operations behaves as intended. A common problem of unit tests is that they're detached from the details of the API's implementation.

Smart Unit Tests enables parameterized unit testing, which unites both techniques. Supported by a test-input generation engine, this methodology combines the client and the implementation perspectives. The functional correctness properties (parameterized unit tests) are checked on most cases of the implementation (test input generation).

A parameterized unit test (PUT) is the straightforward generalization of a unit test through the use of parameters. A PUT makes statements about the code's behavior for an entire set of possible input values, instead of just a single exemplary input value. It expresses assumptions on test inputs, performs a sequence of actions, and asserts properties that should hold in the final state; that is, it serves as the specification. Such a specification doesn't require or introduce any new language or artifact. It's written at the level of the actual APIs implemented by the software product, and in the programming language of the software product. Designers can use them to express intended behavior of the software APIs, developers can use them to drive automated developer testing, and testers can leverage them for in-depth automatic test generation. For example, the following PUT asserts that after adding an element to a non-null list, the element is indeed contained in the list:

```
void TestAdd(ArrayList list, object element)
{
  PexAssume.IsNotNull(list);
  list.Add(element);
  PexAssert.IsTrue(list.Contains(element));
}
```

PUTs separate the following two concerns:

1. The specification of the correctness properties of the code under test for all possible test arguments.
2. The actual "closed" test cases with the concrete arguments.

The engine emits stubs for the first concern, and you're encouraged to flesh them out based on your domain knowledge. Subsequent invocations of Smart Unit Tests will automatically generate and update individual closed test cases.

## Application

Software development teams may be entrenched in various methodologies already, and it's unrealistic to expect them to embrace a new one overnight. Indeed, Smart Unit Tests is not meant as a replacement for any testing practice teams might be following; rather, it's meant to augment any existing practices. Adoption is likely to begin with a gradual embrace, with teams leveraging the default automatic test generation and maintenance capabilities first, and then moving on to write the specifications in code.

**Testing Observed Behavior** Imagine having to make changes to a body of code with no test coverage. You might want to pin down its behavior in terms of a unit test suite before starting, but that's easier said than done:

- The code (product code) might not lend itself to being unit testable. It might have tight dependencies with the external environment that will need to be isolated, and if you can't spot them, you might not even know where to start.
- The quality of the tests might also be an issue, and there are many measures of quality. There is the measure of coverage—how many branches, or code paths, or other program artifacts, in the product code do the tests touch? There's the measure

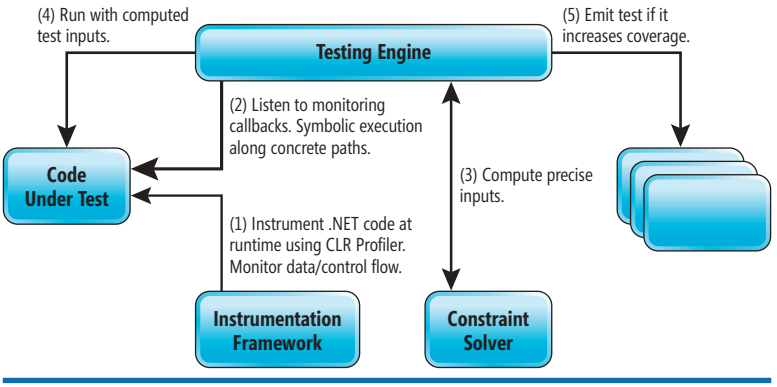



Figure 2 **How Test Case Generation Works Under the Hood**

of assertions that express if the code is doing the right thing. Neither of these measures by themselves is sufficient, however. Instead, what would be nice is a high density of assertions being validated with high code coverage. But it's not easy to do this kind of quality analyses in your head as you write the tests and, as a consequence, you might end up with tests that exercise the same code paths repeatedly; perhaps just testing the "happy path," and you'll never know if the product code can even cope with all those edge cases.

- And, frustratingly, you might not even know what assertions to put in. Imagine being called upon to make changes to an unfamiliar code base!

The automatic test generation capability of Smart Unit Tests is especially useful in this situation. You can baseline the current observed behavior of your code as a suite of tests for use as a regression suite.

**Specification-Based Testing** Software teams can use PUTs as the specification to drive exhaustive test case generation to uncover violations of test assertions. Being freed of much of the manual work necessary to write test cases that achieve high code coverage, the teams can concentrate on tasks that Smart Unit Tests can't automate, such as writing more interesting scenarios as PUTs, and developing integration tests that go beyond the scope of PUTs.

**Automatic Bug Finding** Assertions expressing correctness properties can be stated in multiple ways: as assert statements, as code contracts and more. The nice thing is that these are all compiled down to branches—an if statement with a then branch and an else branch representing the outcome of the predicate being asserted. Because Smart Unit Tests computes inputs that exercise all branches, it becomes an effective bug-finding tool, as well—any input it comes up with that can trigger the else branch represents a bug in the code under test. Thus, all bugs that are reported are actual bugs.

**Reduced Test Case Maintenance** In the presence of PUTs, a significantly lower number of tests cases need to be maintained. In a world where individual closed test cases were written manually, what would happen when the code under test evolved? You'd have to adapt the code of all tests individually, which could represent a significant cost. But by writing PUTs instead, only the PUTs need to be maintained. Then, Smart Unit Tests can automatically regenerate the individual test cases.

## Challenges

**Tool Limitations** The technique of using white-box code analyses with constraint solving works very well on unit-level code that's well isolated. However, the testing engine does have some limitations:

- **Language:** In principle, the testing engine can analyze arbitrary .NET programs, written in any .NET language. However, the test code is generated only in C#.
- **Non-determinism:** The testing engine assumes the code under test is deterministic. If not, it will prune nondeterministic execution paths, or it might go in cycles until it hits exploration bounds.
- **Concurrency:** The testing engine does not handle multithreaded programs.
- **Native code or .NET code that's not instrumented:** The testing engine does not understand native code, that is, x86 instructions called through the Platform Invoke (P/Invoke) feature of the Microsoft .NET Framework. The testing engine doesn't know how to translate such calls into constraints that can be solved by a constraint solver. And even for .NET code, the engine can only analyze code it instruments.
- **Floating point arithmetic:** The testing engine uses an automatic constraint solver to determine which values are relevant for the test case and the code under test. However, the abilities of the constraint solver are limited. In particular, it can't reason precisely about floating point arithmetic.

In these cases the testing engine alerts the developer by emitting a warning, and the engine's behavior in the presence of such limitations can be controlled using custom attributes.

**Writing Good Parameterized Unit Tests** Writing good PUTs can be challenging. There are two core questions to answer:

- **Coverage:** What are good scenarios (sequences of method calls) to exercise the code under test?
- **Verification:** What are good assertions that can be stated easily without reimplementing the algorithm?

A PUT is useful only if it provides answers for both questions.

- Without sufficient coverage; that is, if the scenario is too narrow to reach all the code under test, the extent of the PUT is limited.
- Without sufficient verification of computed results; that is, if the PUT doesn't contain enough assertions, it can't check that the code is doing the right thing. All the PUT does then is check that the code under test doesn't crash or have runtime errors.

In traditional unit testing, the set of questions includes one more: What are relevant test inputs? With PUTs, this question is taken care of by the tooling. However, the problem of finding good assertions is easier in traditional unit testing: The assertions tend to be simpler, because they're written for particular test inputs.

## Wrapping Up

The Smart Unit Tests feature in Visual Studio 2015 Preview lets you specify the intended behavior of the software in terms of its source code, and it uses automated white-box code analysis in conjunction with a constraint solver to generate and maintain a compact suite of relevant tests with high coverage for your .NET code. The benefits span functions—designers can use them to specify the intended behavior of software APIs; developers can use them to drive automated developer testing; and testers can leverage them for in-depth automatic test generation.

The ever-shortening release cycles in software development is driving much of the activities related to planning, specification, implementation and testing to continually happen. This hectic world is challenging us to reevaluate existing practices around those activities. Short, fast, iterative release cycles require taking the collaboration among these functions to a new level. Features such as Smart Unit Tests can help software development teams more easily reach such levels. ■

**Pratap Lakshman** *works in the Developer Division at Microsoft where he is currently a senior program manager on the Visual Studio team, working on testing tools.*

**Thanks** to the following Microsoft technical expert for reviewing this article: *Nikolai Tillmann*


## Work With Relational Data, Not Complex APIs or Services

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

## Give RSSBus a try today and see what mean:

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

rssbus

INTEGRATION YOUR WAY

Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS

vslive.com/lasvegas

Las Vegas MARCH 16 - 20

BALLY'S HOTEL & CASINO LAS VEGAS, NV


LAS VEGAS Code Trip

NAVIGATE THE .NET HIGHWAY

# Code on the Strip

**Visual Studio Live!'s** first stop on its 2015 Code Trip is Las Vegas, situated fittingly near Historic Route 66. Developers, software architects, engineers, and designers will cruise onto the Strip for five days of unbiased and cutting-edge education on the Microsoft Platform. Navigate the .NET Highway with industry experts and Microsoft insiders in 90+ sessions and fun networking events – all designed to make you better at your job.

EVENT PARTNERS Magenic Microsoft

PLATINUM SPONSOR amazon web services

SUPPORTED BY msdn magazine Visual Studio Visual Studio MAGAZINE

PRODUCED BY 1105 MEDIA

# NAVIGATE THE .NET HIGHWAY

## TRACKS INCLUDE:

- Visual Studio / .NET Framework
- JavaScript / HTML5 Client
- Modern App Development
- ASP.NET
- Cross-Platform Mobile Development
- Database and Analytics
- Windows Client
- Cloud Computing
- Windows Phone
- SharePoint Cloud Business Apps

## *Register NOW and Save $300!*

Use promo code VSLFEB4 by February 25

CONNECT WITH VISUAL STUDIO LIVE!

**twitter.com/vslive – @VSLive**

**facebook.com – Search "VSLive"**

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

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

TURN THE PAGE FOR MORE EVENT DETAILS

# Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

vslive.com/lasvegas

**Las Vegas** MARCH 16 - 20

BALLY'S HOTEL & CASINO LAS VEGAS, NV

**Bally's Hotel & Casino** will play host to Visual Studio Live!, and is offering a special reduced room rate to conference attendees.


CONNECT WITH VISUAL STUDIO LIVE!

**twitter.com/vslive – @VSLive**

**facebook.com – Search "VSLive"**

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

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

# AGENDA AT-A-GLANCE

| ASP.NET | Cloud Computing | Cross-Platform Mobile Development | JavaScript / HTML5 Client |
|---|---|---|---|

| START TIME | END TIME | Visual Studio Live! Pre-Conference Workshops: Monday, March 16, 2015 |
|---|---|---|
| 7:30 AM | 9:00 AM | Pre-Conference Workshop Registration - Coffee and Morning Pastries |
| 9:00 AM | 6:00 PM | **M01** - Workshop: Native Mobile App Development for iOS, Android and Windows Using C# - *Marcel de Vries & Rockford Lhotka* |
| 6:45 PM | 9:00 PM | Dine-A-Round |

| START TIME | END TIME | Visual Studio Live! Day 1: Tuesday, March 17, 2015 | |
|---|---|---|---|
| 7:00 AM | 8:00 AM | Registration - Coffee and Morning Pastries | |
| 8:00 AM | 9:00 AM | Keynote: To Be Announced | |
| 9:15 AM | 10:30 AM | **T01** - An Overview of the Xamarin Programming Platforms - *Laurent Bugnion* | **T02** - JavaScript for the C# (and Java) Developer - *Philip Japikse* |
| 10:45 AM | 12:00 PM | **T07** - Building Cross-Platform Applications with XAML, Xamarin, Xamarin.Forms and MVVM Light - *Laurent Bugnion* | **T08** - Write Next Generation, Object Oriented JavaScript, with TypeScript - *Rachel Appel* |
| 12:00 PM | 1:30 PM | Lunch - Visit Exhibitors | |
| 1:30 PM | 2:45 PM | **T13** - Microsoft - To Be Announced | **T14** - It's the Top 10 of Cool HTML5 Features Every Developer Should Know Right Now - *Gill Cleeren* |
| 3:00 PM | 4:15 PM | **T19** - Creating Responsive Cross-Platform Native/ Web Apps with JavaScript and Bootstrap - *Ben Dewey* | **T20** - I Just Met You, and "This" is Crazy, But Here's My NaN, So Call (me) Maybe? - *Rachel Appel* |
| 4:15 PM | 5:30 PM | Welcome Reception | |

| START TIME | END TIME | Visual Studio Live! Day 2: Wednesday, March 18, 2015 | |
|---|---|---|---|
| 7:00 AM | 8:00 AM | Registration - Coffee and Morning Pastries | |
| 8:00 AM | 9:00 AM | Keynote: To Be Announced - *Rockford Lhotka, CTO, Magenic* | |
| 9:15 AM | 10:30 AM | **W01** - Cross-Platform Mobile Apps Using AngularJS and the Ionic Framework - *Ben Dewey* | **W02** - Getting Started with jQuery and How it Works Together with ASP.NET WebForms and MVC - *Gill Cleeren* |
| 10:45 AM | 12:00 PM | **W07** - Building Cross-Platform Apps in C# - *Rockford Lhotka* | **W08** - Build Data-Centric HTML5 Single Page Applications with Breeze - *Brian Noyes* |
| 12:00 PM | 1:30 PM | Birds-of-a-Feather Lunch - Visit Exhibitors | |
| 1:30 PM | 2:45 PM | **W13** - Building Your First Universal Application for Windows Phone and Windows Store - *Brian Peek* | **W14** - Busy Developer's Guide to MEANJS - *Ted Neward* |
| 3:00 PM | 4:15 PM | **W19** - Getting Started with Windows Phone 8.1 Development - *Nick Landry* | **W20** - Creating Angular Applications Using Visual Studio LightSwitch - *Michael Washington* |
| 4:30 PM | 5:45 PM | **W25** - Strike Up a Conversation with Cortana on Windows Phone - *Walt Ritscher* | **W26** - JavaScript Patterns for the C# Developer - *Ben Hoelting* |
| 7:00 PM | 9:00 PM | Visual Studio Live! Evening Event sponsored by amazon web services | |

| START TIME | END TIME | Visual Studio Live! Day 3: Thursday, March 19, 2015 | |
|---|---|---|---|
| 7:00 AM | 8:00 AM | Registration - Coffee and Morning Pastries | |
| 8:00 AM | 9:15 AM | **TH01** - From WPF to Universal Apps: A Guide Forward - *Rockford Lhotka* | **TH02** - AngularJS for ASP.NET MVC Developers - *Miguel Castro* |
| 9:30 AM | 10:45 AM | **TH07** - UX Design Principle Fundamentals for Non-Designers - *Billy Hollis* | **TH08** - Securing Angular Apps - *Brian Noyes* |
| 11:00 AM | 12:15 PM | **TH13** - XAML: Achieving Your Moment of Clarity - *Miguel Castro* | **TH14** - Unit Testing JavaScript Applications - *Ben Dewey* |
| 12:15 PM | 1:30 PM | Lunch | |
| 1:30 PM | 2:45 PM | **TH19** - Tricks and Shortcuts for Amazing UI in XAML - *Billy Hollis* | **TH20** - Building Single Page Web Applications Using JavaScript and the MVVM Pattern - *Ben Hoelting* |
| 3:00 PM | 4:15 PM | **TH25** - WPF Data Binding in Depth - *Brian Noyes* | **TH26** - Busy Developer's Guide to SignalR and WebSockets - *Ted Neward* |

| START TIME | END TIME | Visual Studio Live! Post-Conference Workshops: Friday, March 20, 2015 |
|---|---|---|
| 7:30 AM | 8:00 AM | Post-Conference Workshop Registration - Coffee and Morning Pastries |
| 8:00 AM | 5:00 PM | **F01** - Workshop: Hybrid Mobile Apps with Cordova, Angular, and Azure - *Brian Noyes* |

*Speakers and sessions subject to change*

# Modern Apps LIVE!

MOBILE, CROSS-DEVICE & CLOUD DEVELOPMENT

Presented in partnership with Magenic

**BONUS CONTENT!** Modern Apps Live! is now a part of Visual Studio Live! Las Vegas at no additional cost!


| SharePoint Cloud Business Apps | Database and Analytics | Visual Studio / .NET Framework | Windows Client | Windows Phone | Modern Apps Live! |
|---|---|---|---|---|---|

(Separate entry fee required)

| | | |
|---|---|---|
| **M02** - Workshop: Deep Dive into Visual Studio 2013, TFS and Visual Studio Online - *Brian Randell* | **M03** - Workshop: UX Design Process Essentials – Steps and Techniques - *Billy Hollis* | **M04** - Workshop: Modern App Technology Overview - Android, iOS, Cloud, and Mobile Web - *Nick Landry, Kevin Ford, & Steve Hughes* |

| | | | |
|---|---|---|---|
| **T03** - What's Up with ASP.NET vNext (Who Moved My Cheese?) - *Mark Michaelis* | **T04** - Cloud or Not, 10 Reasons Why You Must Know "Web Sites" - *Vishwas Lele* | **T05** - New IDE and Editor Features in Visual Studio - *Deborah Kurata* | **T06** - Defining Modern App Development - *Rockford Lhotka* |
| **T09** - Moving DevOps to the Cloud Using Visual Studio Online, Release Management Online, and Azure - *Donovan Brown* | **T10** - Azure 10-10: Top 10 Azure Announcements in "T-10" Months - *Vishwas Lele* | **T11** - Visual Studio Online: An Update Every Three Weeks - *Brian Randell* | **T12** - Modern App Architecture - *Rockford Lhotka* |
| **T15** - ASP.NET MVC For The WebForms Developer - *Philip Japikse* | **T16** - Busy Developer's Guide to NoSQL - *Ted Neward* | **T17** - Defensive Coding Techniques in C# - *Deborah Kurata* | **T18** - ALM with Visual Studio Online (TFS) and Git - *Brian Randell* |
| **T21** - Slice Development Time With ASP.NET MVC, Visual Studio, and Razor - *Philip Japikse* | **T22** - Busy Developer's Guide to Six Developer Cloud Platforms - *Ted Neward* | **T23** - Automated Cross Browser Testing of Your Web Applications with Visual Studio CodedUI - *Marcel de Vries* | **T24** - Reusing Logic Across Platforms - *Kevin Ford* |

| | | | |
|---|---|---|---|
| **W03** - ASP.NET vNext in All Its Glory - *Adam Tuliper* | **W04** - T-SQL for Application Developers - Attendees Choose - *Kevin Goff* | **W05** - Using Microsoft Application Insights to Implement a Build, Measure, Learn Loop - *Marcel de Vries* | **W06** - Coding for Quality and Maintainability - *Jason Bock* |
| **W09** - Understanding the Power of ASP.NET MVC vNext - *Ben Hoelting* | **W10** - SQL Server Reporting Services for Application Developers - Attendees Choose - *Kevin Goff* | **W11** - Build It and Ship It with TFS and Release Management - *Brian Randell* | **W12** - Start Thinking Like a Designer - *Anthony Handley* |
| **W15** - Improving Web Application Performance - *Ben Hoelting* | **W16** - A Tour of Azure for Developers - *Adam Tuliper* | **W17** - Programming C# 6.0 - *Mark Michaelis* | **W18** - Applied UX: iOS, Android, Windows - *Anthony Handley* |
| **W21** - To Be Announced | **W22** - 10 Ways to Guarantee Your Azure Project Will Fail! - *Michael Collier* | **W23** - Building Games for Windows and Windows Phone Using Unity and Other Frameworks - *Brian Peek* | **W24** - Leveraging Azure Services - *Kevin Ford* |
| **W27** - WCF & Web API: Can We All Just Get Along?!? - *Miguel Castro* | **W28** - Intro to Azure API Management - *Michael Collier* | **W29** - The Busy Developers Guide to Virtualization with Hyper-V - *Brian Randell* | **W30** - Advanced Modern App Architecture Concepts - *Marcel de Vries* |

| | | | |
|---|---|---|---|
| **TH03** - Fiddler and Your Website - *Robert Boedigheimer* | **TH04** - SQL Server in Azure Virtual Machines - *Eric D. Boyd* | **TH05** - Managing the .NET Compiler - *Jason Bock* | **TH06** - Building a Modern App with Xamarin - *Nick Landry* |
| **TH09** - Web Essentials - *Robert Boedigheimer* | **TH10** - Microsoft Azure SQL Database - SQL Server in the Cloud - *Leonard Lobel* | **TH11** - Real World Scrum with Team Foundation Server 2013 & Visual Studio Online - *Benjamin Day* | **TH12** - Building an iOS and Android App with Xamarin.Forms - *Brent Edwards* |
| **TH15** - Node.js for .NET Developers - *Jason Bock* | **TH16** - Database Development with SQL Server Data Tools - *Leonard Lobel* | **TH17** - Git for the Microsoft Developer - *Eric D. Boyd* | **TH18** - Building for the Modern Web with Single Page Applications - *Allen Conway* |
| **TH21** - Building Rich UI with ASP.NET MVC and Bootstrap - *Walt Ritscher* | **TH22** - Creating Visual Studio Cloud Business Applications for SharePoint 2013 Using AngularJS - *Michael Washington* | **TH23** - Getting Started with AWS for .NET Developers - *Norm Johanson* | **TH24** - Building a Windows App - *Brent Edwards* |
| **TH27** - ASP.NET - Rudiments of Routing - *Walt Ritscher* | **TH28** - Cloud Business Apps and SharePoint Workflows - *Michael Washington* | **TH29** - Zero to Hero: Untested to Tested with Visual Studio Fakes - *Benjamin Day* | **TH30** - Analyzing Results with Power BI - *Steve Hughes* |

(Separate entry fee required)

| | | |
|---|---|---|
| **F02** - Workshop: SQL Server 2014 for Developers - *Leonard Lobel* | **F03** - *Workshop: Developing Neural Networks using C#* - *James McCaffrey* | **F04** - Workshop: Modern App Development In-Depth - iOS, Android, Windows, and Web - *Brent Edwards, Anthony Handley, & Allen Conway* |

# What Every Programmer Should Know About Compiler Optimizations

Hadi Brais

High-level programming languages offer many abstract programming constructs such as functions, conditional statements and loops that make us amazingly productive. However, one disadvantage of writing code in a high-level programming language is the potentially significant decrease in performance. Ideally, you should write understandable, maintainable code—without compromising performance. For this reason, compilers attempt to automatically optimize the code to improve its performance, and they've become quite sophisticated in doing so nowadays. They can transform loops, conditional statements, and recursive functions; eliminate whole blocks of code; and take advantage of the target instruction set architecture (ISA) to make the code fast and compact. It's much better to focus on writing understandable code, than making manual optimizations that result in cryptic, hard-to-maintain code. In fact, manually optimizing the code might prevent the compiler from performing additional or more efficient optimizations.

Rather than manually optimizing code, you should consider aspects of your design, such as using faster algorithms, incorporating thread-level parallelism and using framework-specific features (such as using move constructors).

This article is about Visual C++ compiler optimizations. I'm going to discuss the most important optimization techniques and the decisions a compiler has to make in order to apply them. The purpose isn't to tell you how to manually optimize the code, but to show you why you can trust the compiler to optimize the code on your behalf. This article is by no means a complete examination of the optimizations performed by the Visual C++ compiler. However, it demonstrates the optimizations you really want to know about and how to communicate with the compiler to apply them.

There are other important optimizations that are currently beyond the capabilities of any compiler—for example, replacing an inefficient algorithm with an efficient one, or changing the layout of a data structure to improve its locality. However, such optimizations are outside the scope of this article.

This article discusses:

- Important Visual C++ compiler optimizations
- How the Visual C++ compiler uses the vector unit of the processor
- How to control compiler optimizations
- Compiler optimizations in the Microsoft .NET Framework

Technologies discussed:

Visual Studio 2013, Visual C++ Compiler, Microsoft .NET Framework

Code download available at:

msdn.microsoft.com/magazine/msdnmag0215

## Defining Compiler Optimizations

An optimization is the process of transforming a piece of code into another functionally equivalent piece of code for the purpose of improving one or more of its characteristics. The two most

important characteristics are the speed and size of the code. Other characteristics include the amount of energy required to execute the code, the time it takes to compile the code and, in case the resulting code requires Just-in-Time (JIT) compilation, the time it takes to JIT compile the code.

Compilers are constantly improving in terms of the techniques they use to optimize the code. However, they're not perfect. Still, instead of spending time manually tweaking a program, it's usually much more fruitful to use specific features provided by the compiler and let the compiler tweak the code.

There are four ways to help the compiler optimize your code more effectively:

1. Write understandable, maintainable code. Don't look at the object-oriented features of Visual C++ as the enemies of performance. The latest version of Visual C++ can keep such overhead to a minimum and sometimes completely eliminate it.
2. Use compiler directives. For example, tell the compiler to use a function-calling convention that's faster than the default one.
3. Use compiler-intrinsic functions. An intrinsic function is a special function whose implementation is provided automatically by the compiler. The compiler has an intimate knowledge of the function and substitutes the function call with an extremely efficient sequence of instructions that take advantage of the target ISA. Currently, the Microsoft .NET Framework doesn't support intrinsic functions, so none of the managed languages support them. However, Visual C++ has extensive support for this feature. Note that while using intrinsic functions can improve the performance of the code, it reduces its readability and portability.
4. Use profile-guided optimization (PGO). With this technique, the compiler knows more about how the code is going to behave at run time and can optimize it accordingly.

The purpose of this article is to show you why you can trust the compiler by demonstrating the optimizations performed on inefficient but understandable code (applying the first method). Also, I'll provide a short introduction to profile-guided optimization and mention some of the compiler directives that enable you to fine-tune some parts of your code.

There are many compiler optimization techniques ranging from simple transformations, such as constant folding, to extreme transformations, such as instruction scheduling. However, in this article, I'll limit discussion to some of the most important optimizations—those that can significantly improve performance (by a double-digit percentage) and reduce code size: function inlining, COMDAT optimizations and loop optimizations. I'll discuss the first two in the next section, then show how you can control the optimizations performed by Visual C++. Finally, I'll take a brief look at optimizations in the .NET Framework. Throughout this article, I'll be using Visual Studio 2013 to build the code.

Figure 1 **The source1.c File**

```
#include <stdio.h> // scanf_s and printf.
#include "Source2.h"

int square(int x) { return x*x; }

main() {
  int n = 5, m;
  scanf_s("%d", &m);
  printf("The square of %d is %d.", n, square(n));
  printf("The square of %d is %d.", m, square(m));
  printf("The cube of %d is %d.", n, cube(n));
  printf("The sum of %d is %d.", n, sum(n));
  printf("The sum of cubes of %d is %d.", n, sumOfCubes(n));
  printf("The %dth prime number is %d.", n, getPrime(n));
}
```

## Link-Time Code Generation

Link-Time Code Generation (LTCG) is a technique for performing whole program optimizations (WPO) on C/C++ code. The C/C++ compiler compiles each source file separately and produces the corresponding object file. This means the compiler can only apply optimizations on a single source file rather than on the whole program. However, some important optimizations can be performed only by looking at the whole program. You can apply these optimizations at link time rather than at compile time because the linker has a complete view of the program.

> One disadvantage of writing code in a high-level programming language is the potentially significant decrease in performance.

When LTCG is enabled (by specifying the /GL compiler switch), the compiler driver (cl.exe) will invoke only the front end of the compiler (c1.dll or c1xx.dll) and postpone the work of the back end (c2.dll) until link time. The resulting object files contain C Intermediate Language (CIL) code rather than machine-dependent assembly code. Then, when the linker (link.exe) is invoked, it sees that the object files contain CIL code and invokes the back end of the compiler, which in turn performs WPO, generates the binary object files, and returns to the linker to stitch all object files together and produce the executable.

The front end actually performs some optimizations, such as constant folding, irrespective of whether optimizations are enabled or disabled. However, all important optimizations are performed by the back end of the compiler and can be controlled using compiler switches.

LTCG enables the back end to perform many optimizations aggressively (by specifying /GL together with the /O1 or /O2 and /Gw compiler switches and the /OPT:REF and /OPT:ICF linker switches). In this article, I'll discuss only function inlining and COMDAT optimizations. For a complete list of LTCG optimizations, refer to the documentation. Note that the linker can perform LTCG on native object files, mixed native/managed object files, pure managed object files, safe managed object files and safe .netmodules.

I'll build a program consisting of two source files (source1.c and source2.c) and a header file (source2.h). The source1.c and source2.c

files are shown in **Figure 1** and **Figure 2**, respectively. The header file, which contains the prototypes of all functions in source2.c, is quite simple, so I won't show it here.

The source1.c file contains two functions: the square function, which takes an integer and returns its square, and the main function of the program. The main function calls the square function and all functions from source2.c except isPrime. The source2.c file contains five functions: the cube function returns the cube of a given integer; the sum function returns the sum of all integers from 1 to a given integer; the sumOfcubes function returns the sum of cubes of all integers from 1 to a given integer; the isPrime function determines whether a given integer is prime; and the getPrime function, which returns the xth prime number. I've omitted error checking because it's not of interest in this article.

The code is simple but useful. There are a number of functions that perform simple computations; some require simple for loops. The getPrime function is the most complex because it contains a while loop and, within the loop, it calls the isPrime function, which also contains a loop. I'll use this code to demonstrate one of the most important compiler optimizations, namely function inlining, and some other optimizations.

I'll build the code under three different configurations and examine the results to determine how it was transformed by the compiler. If you follow along, you'll need the assembler output file (produced with the /FA[s] compiler switch) to examine the resulting assembly code, and the map file (produced with the /MAP linker switch) to determine the COMDAT optimizations that have been performed (the linker can also report this if you use the /verbose:icf and /verbose:ref switches). So make sure these switches are specified in all of the following configurations I discuss. Also, I'll be using the C compiler (/TC) so that the generated code is easier to examine. However, everything I discuss here also applies to C++ code.

Figure 2 **The source2.c File**

```
#include <math.h> // sqrt.
#include <stdbool.h> // bool, true and false.
#include "Source2.h"

int cube(int x) { return x*x*x; }

int sum(int x) {
  int result = 0;
  for (int i = 1; i <= x; ++i) result += i;
  return result;
}

int sumOfCubes(int x) {
  int result = 0;
  for (int i = 1; i <= x; ++i) result += cube(i);
  return result;
}

static
bool isPrime(int x) {
  for (int i = 2; i <= (int)sqrt(x); ++i) {
    if (x % i == 0) return false;
  }
  return true;
}

int getPrime(int x) {
  int count = 0;
  int candidate = 2;
  while (count != x) {
    if (isPrime(candidate))
      ++count;
  }
  return candidate;
}
```

## The Debug Configuration

The Debug configuration is used mainly because all back-end optimizations are disabled when you specify the /Od compiler switch without specifying the /GL switch. When building the code under this configuration, the resulting object files will contain binary code that corresponds exactly to the source code. You can examine the resulting assembler output files and the map file to confirm this. This configuration is equivalent to the Debug configuration of Visual Studio.

## The Compile-Time Code Generation Release Configuration

This configuration is similar to the Release configuration in which optimizations are enabled (by specifying the /O1, /O2 or /Ox compile switches), but without specifying the /GL compiler switch. Under this configuration, the resulting object files will contain optimized binary code. However, no optimizations at the whole-program level are performed.

By examining the generated assembly listing file of source1.c, you'll notice that two optimizations have been performed. First, the first call to the square function, square(n), in **Figure 1** has been completely eliminated by evaluating the computation at compile time. How did this happen? The compiler determined that the square function is small, so it should be inlined. After inlining it, the compiler determined that the value of the local variable n is known and doesn't change between the assignment statement and the function call. Therefore, it concluded that it's safe to execute the multiplication and substitute the result (25). In the second optimization, the second call to the square function, square(m), has been inlined, as well. However, because the value of m isn't known at compile time, the compiler can't evaluate the computation, so the actual code is emitted.

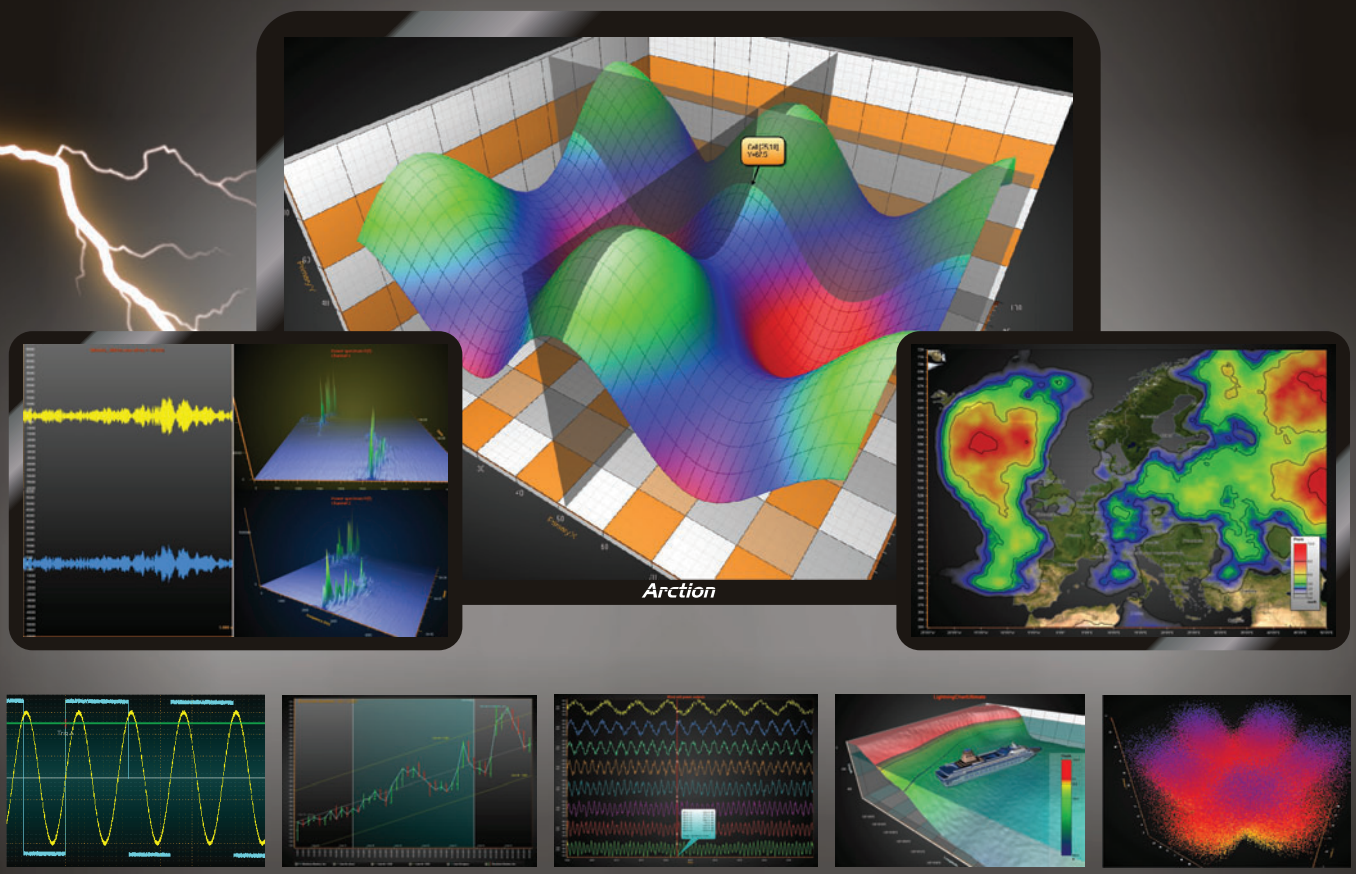
Now I'll examine the assembly listing file of source2.c, which is much more interesting. The call to the cube function in sumOfCubes has been inlined. This in turn has enabled the compiler to perform significant optimizations on the loop (as you'll see in the "Loop Optimizations" section). In addition, the SSE2 instruction set is being used in the isPrime function to convert from int to double when calling the sqrt function and also to convert from double to int when returning from sqrt. And sqrt is called only once before the loop starts. Note that if no /arch switch is specified to the compiler, the x86 compiler uses SSE2 by default. Most deployed x86 processors, as well as all x86-64 processors, support SSE2.

## The Link-Time Code Generation Release Configuration

The LTCG Release configuration is identical to the Release configuration in Visual Studio. In this configuration, optimizations are enabled and the /GL compiler switch is specified. This switch is implicitly specified when using /O1 or /O2. It tells the compiler to emit CIL object files rather than assembly object files. In this way, the linker invokes the back end of the compiler to perform WPO

The FASTEST rendering Data Visualization components for WPF and WinForms...

# LightningChart




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

**WPF charts performance comparison**

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

**WinForms charts performance comparison**

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

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

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

Download a free 30-day evaluation from

**www.LightningChart.com**

Arction

as described earlier. Now I'll discuss several WPO optimizations to show the immense benefit of LTCG. The assembly code listings generated with this configuration are available online.

As long as function inlining is enabled (/Ob, which is turned on whenever you request optimizations), the /GL switch enables the compiler to inline functions defined in other translation units irrespective of whether the /Gy compiler switch (discussed a bit later) is specified. The /LTCG linker switch is optional and provides guidance for the linker only.

> Ideally, developers prefer to write understandable, maintainable code—without compromising performance.

By examining the assembly listing file of source1.c, you can see that all function calls except for scanf_s have been inlined. As a result, the compiler was able to execute the computations of the cube, sum and sumOfCubes. Only the isPrime function hasn't been inlined. However, if it has been inlined manually in getPrime, the compiler would still inline getPrime in main.

As you can see, function inlining is important not only because it optimizes away a function call, but also because it enables the compiler to perform many other optimizations as a result. Inlining a function usually improves performance at the expense of increasing the code size. Excessive use of this optimization leads to a phenomenon known as code bloat. At every call site, the compiler performs a cost/benefit analysis and then decides whether to inline the function.

Due to the importance of inlining, the Visual C++ compiler provides much more support than what the standard dictates regarding inlining control. You can tell the compiler to never inline a range of functions by using the auto_inline pragma. You can tell the compiler to never inline a specific function or method by marking it with __declspec(noinline). You can mark a function with the inline keyword to give a hint to the compiler to inline the function (although the compiler may choose to ignore this hint if inlining would be a net loss). The inline keyword has been available since the first version of C++—it was introduced in C99. You can use the Microsoft-specific keyword __inline in both C and C++ code; it's useful when you're using an old version of C that doesn't support this keyword. Furthermore, you can use the __forceinline keyword (C and C++) to force the compiler to always inline a function whenever possible. And last, but not least, you can tell the compiler to unfold a recursive function either to a specific or indefinite depth by inlining it using the inline_recursion pragma. Note that the compiler currently offers no features that enable you to control inlining at the call site rather than at the function definition.

The /Ob0 switch disables inlining completely, which takes effect by default. You should use this switch when debugging (it's automatically specified in the Visual Studio Debug Configuration). The /Ob1 switch tells the compiler to only consider functions for inlining that are marked with inline, __inline or __forceinline. The /Ob2 switch, which takes effect when specifying /O[1|2|x], tells the compiler to consider any function for inlining. In my opinion, the only reason to use the inline or __inline keywords is to control inlining with the /Ob1 switch.

The compiler won't be able to inline a function in certain conditions. One example is when calling a virtual function virtually; the function can't be inlined because the compiler may not know which function is going to be called. Another example is when calling a function through a pointer to the function rather than using its name. You should strive to avoid such conditions to enable inlining. Refer to the MSDN documentation for a complete list of such conditions.

Function inlining isn't the only optimization that's more effective when applied at the whole program level. In fact, most optimizations become more effective at that level. In the rest of this section, I'll discuss a specific class of such optimizations called COMDAT optimizations.

By default, when compiling a translation unit, all code will be stored in a single section in the resulting object file. The linker operates at the section level. That is, it can remove sections, combine sections, and reorder sections. This precludes the linker from performing three optimizations that can significantly (double-digit percentage) reduce the size of the executable and improve its performance. The first is eliminating unreferenced functions and global variables. The second is folding identical functions and constant global variables. The third is reordering functions and global variables so those functions that fall on the same execution path and those variables that are accessed together are physically located closer in memory to improve locality.

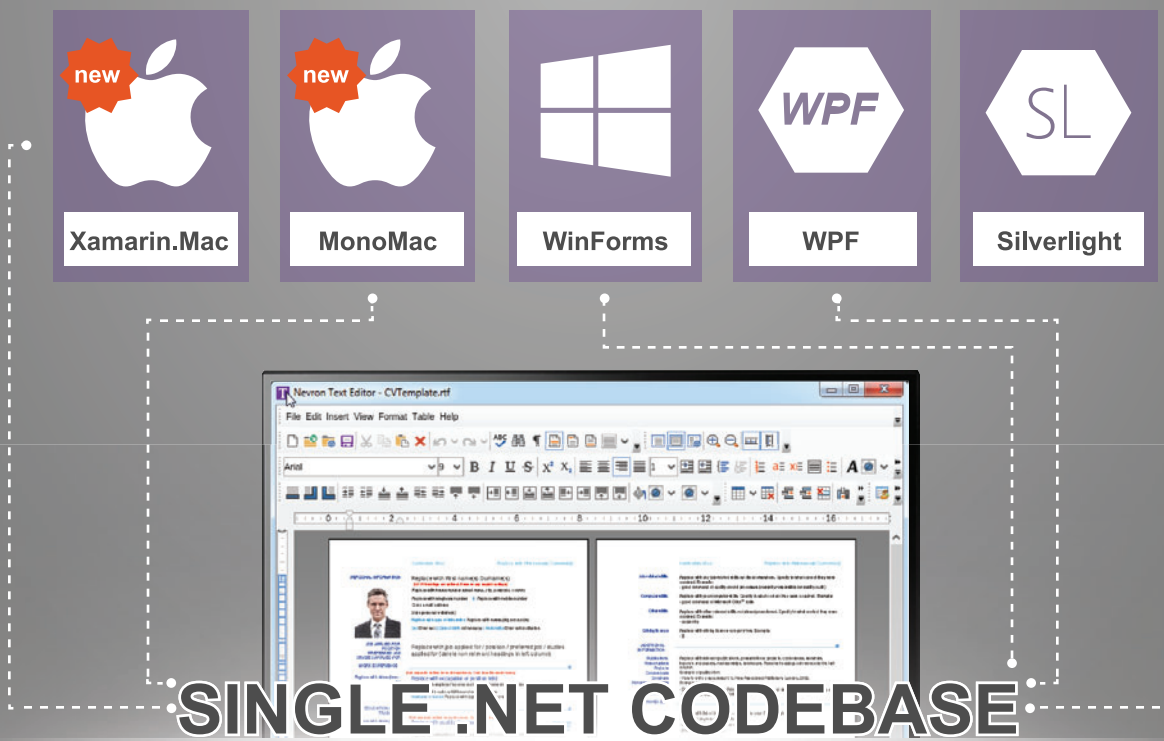
To enable these linker optimizations, you can tell the compiler to package functions and variables into separate sections by specifying the /Gy (function-level linking) and /Gw (global data optimization) compiler switches, respectively. Such sections are called COMDATs. You can also mark a particular global data variable with __declspec(selectany) to tell the compiler to pack the variable into a COMDAT. Then, by specifying the /OPT:REF linker switch, the linker will eliminate unreferenced functions and global variables. Also, by specifying the /OPT:ICF switch, the linker will fold identical functions and global constant variables. (ICF stands for Identical COMDAT Folding.) With the /ORDER linker switch, you can instruct the linker to place COMDATs into the resulting image in a specific order. Note that all of these optimizations are linker optimizations and don't require the /GL compiler switch. The /OPT:REF and /OPT:ICF switches should be disabled while debugging for obvious reasons.

You should use LTCG whenever possible. The only reason not to use LTCG is when you want to distribute the resulting object and library files. Recall that these files contain CIL code rather than assembly code. CIL code can be consumed only by the compiler/linker of the same version that produced it, which can significantly limit the usability of the object files because developers have to have the same version of the compiler to use these files. In this case, unless you're willing to distribute the object files for every compiler version, you should use compile-time code generation instead. In addition to limited usability, these

NEVRON
# OPEN VISION for .NET

www.nevron.com

# Write Once, Run Everywhere




Nevron Open Vision is the first suite of .NET controls for building cross-platform user interfaces and applications from a single code base.

## Nevron Open Vision includes:

- NOV **WIDGETS** for .NET
- NOV **TEXT EDITOR** for .NET
- NOV **GAUGE** for .NET
- NOV **BARCODE** for .NET

COMING SOON:

- NOV **CHART** for .NET
- NOV **DIAGRAM** for .NET
- NOV **GRID** for .NET

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

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

Visual Studio Partner

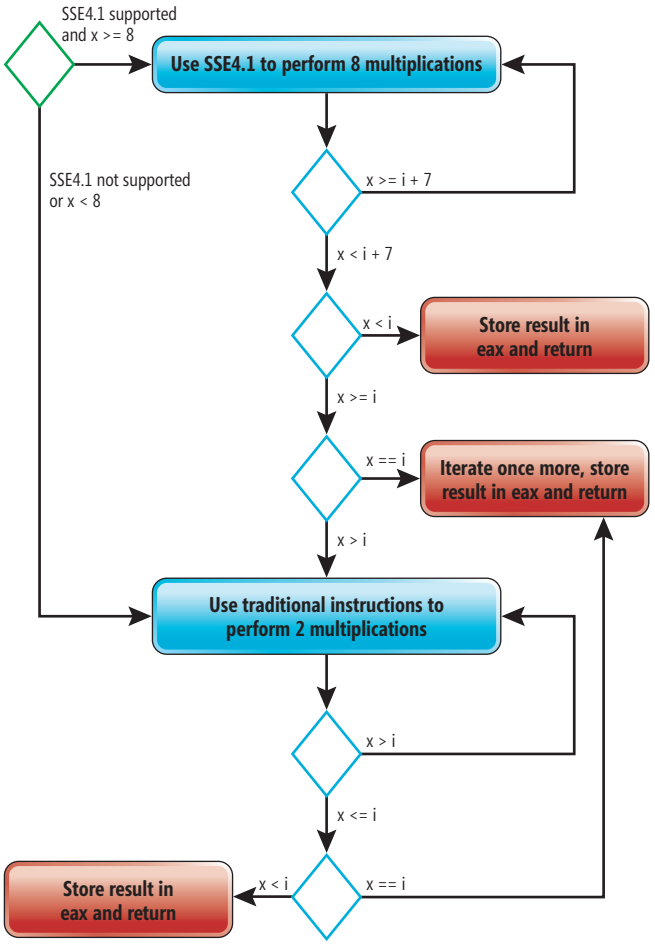



Figure 3 **Control Flow Graph of sumOfCubes**

object files are many times larger in size than the corresponding assembler object files. However, do keep in mind the huge benefit of CIL object files, which is enabling WPO.

## Loop Optimizations

The Visual C++ compiler supports several loop optimizations, but I'll discuss only three: loop unrolling, automatic vectorization and loop-invariant code motion. If you modify the code in **Figure 1** so that m is passed to sumOfCubes instead of n, the compiler won't be able to determine the value of the parameter, so it must compile the function to handle any argument. The resulting function is highly optimized and its size is rather large, so the compiler won't inline it.

Compiling the code with the /O1 switch results in assembly code that's optimized for space. In this case, no optimizations will be performed on the sumOfCubes function. Compiling with the /O2 switch results in code that's optimized for speed. The size of the code will be significantly larger yet significantly faster because the loop inside sumOfCubes has been unrolled and vectorized. It's important to understand that vectorization would not be possible without inlining the cube function. Moreover, loop unrolling would not be that effective without inlining. A simplified graphical representation of the resulting assembly code is shown in **Figure 3**. The flow graph is the same for both x86 and x86-64 architectures.

In **Figure 3**, the green diamond is the entry point and the red rectangles are the exit points. The blue diamonds represent conditions that are being executed as part of the sumOfCubes function at run time. If SSE4 is supported by the processor and x is larger than or equal to eight, then SSE4 instructions will be used to perform four multiplications at the same time. The process of executing the same operation on multiple values simultaneously is called vectorization. Also, the compiler will unroll the loop twice; that is, the loop body will be repeated twice in every iteration. The combined effect is that eight multiplications will be performed for every iteration. When x becomes less than eight, traditional instructions will be used to execute the rest of the computations. Note that the compiler has emitted three exit points containing separate epilogues in the function instead of just one. This reduces the number of jumps.

Loop unrolling is the process of repeating the loop body within the loop so that more than one iteration of the loop is executed within a single iteration of the unrolled loop. The reason this improves performance is that loop control instructions will be executed less frequently. Perhaps more important, it might enable the compiler to perform many other optimizations, such as vectorization. The downside of unrolling is that it increases the code size and register pressure. However, depending on the loop body, it might improve performance by a double-digit percentage.

Unlike x86 processors, all x86-64 processors support SSE2. Moreover, you can take advantage of the AVX/AVX2 instruction sets of the latest x86-64 microarchitectures from Intel and AMD by specifying the /arch switch. Specifying /arch:AVX2 enables the compiler to use the FMA and BMI instruction sets, as well.

Currently, the Visual C++ compiler doesn't enable you to control loop unrolling. However, you can emulate this technique by using templates together with the __ forceinline keyword. You can disable auto-vectorization on a specific loop using the loop pragma with the no_vector option.

By looking at the generated assembly code, keen eyes would notice that the code can be optimized a bit more. However, the compiler has done a great job already and won't spend much more time analyzing the code and applying minor optimizations.

someOfCubes is not the only function whose loop has been unrolled. If you modify the code so that m is passed to the sum function instead of n, the compiler won't be able to evaluate the function and, therefore, it has to emit its code. In this case, the loop will be unrolled twice.

The last optimization I'll discuss is loop-invariant code motion. Consider the following piece of code:

```
int sum(int x) {
  int result = 0;
  int count = 0;
  for (int i = 1; i <= x; ++i) {
    ++count;
    result += i;
  }
  printf("%d", count);
  return result;
}
```

The only change here is that I have an additional variable that's being incremented in each iteration and then printed. It's not hard to see that this code can be optimized by moving the increment of the count variable outside the loop. That is, I can just assign x

to the count variable. This optimization is called loop-invariant code motion. The loop-invariant part clearly indicates that this technique only works when the code doesn't depend on any of the expressions in the loop header.

Now here's the catch: If you apply this optimization manually, the resulting code might exhibit degraded performance in certain conditions. Can you see why? Consider what happens when x is nonpositive. The loop never executes, which means that in the unoptimized version, the variable count won't be touched. However, in the manually optimized version, an unnecessary assignment from x to count is executed outside the loop! Moreover, if x was negative, then count would hold the wrong value. Both humans and compilers are susceptible to such pitfalls. Fortunately, the Visual C++ compiler is smart enough to realize this by emitting the condition of the loop before the assignment, resulting in an improved performance for all values of x.

> Some important optimizations can be performed only by looking at the whole program.

In summary, if you are neither a compiler nor a compiler optimizations expert, you should avoid making manual transformations to your code just to make it look faster. Keep your hands clean and trust the compiler to optimize your code.

## Controlling Optimizations

In addition to the compiler switches /O1, /O2 and /Ox, you can control optimizations for specific functions using the optimize pragma, which looks like this:

```
#pragma optimize( "[optimization-list]", {on | off} )
```

The optimization list can be either empty or contain one or more of the following values: g, s, t and y. These correspond to the compiler switches /Og, /Os, /Ot and /Oy, respectively.

An empty list with the off parameter causes all of these optimizations to be turned off regardless of the compiler switches that have been specified. An empty list with the on parameter causes the specified compiler switches to take effect.

The /Og switch enables global optimizations, which are those that can be performed by looking at the function being optimized only, not at any of the functions that it calls. If LTCG is enabled, /Og enables WPO.

The optimize pragma is useful when you want different functions to be optimized in different ways—some for space and others for speed. However, if you really want to have that level of control, you should consider profile-guided optimization (PGO), which is the process of optimizing the code by using a profile that contains behavioral information recorded while running an instrumented version of the code. The compiler uses the profile to make better decisions on how to optimize the code. Visual Studio provides the necessary tools to apply this technique on native and managed code.

## Optimizations in .NET

There's no linker involved in the .NET compilation model. However, there is a source code compiler (C# compiler) and a JIT compiler. The source code compiler performs only minor optimizations. For example, it doesn't perform function inlining and loop optimizations. Instead, these optimizations are handled by the JIT compiler. The JIT compiler that ships with all versions of the .NET Framework up to 4.5 doesn't support SIMD instructions. However, the JIT compiler that ships with the .NET Framework 4.5.1 and later versions, called RyuJIT, supports SIMD.

What's the difference between RyuJIT and Visual C++ in terms of optimization capabilities? Because it does its work at run time, RyuJIT can perform optimizations that Visual C++ can't. For example, at run time, RyuJIT might be able to determine that the condition of an if statement is never true in this particular run of the application and, therefore, it can be optimized away. Also RyuJIT can take advantage of the capabilities of the processor on which it's running. For example, if the processor supports SSE4.1, the JIT compiler will only emit SSE4.1 instructions for the sumOfcubes function, making the generated code much more compact. However, it can't spend much time optimizing the code because the time taken to JIT-compile impacts the performance of the application. On the other hand, the Visual C++ compiler can spend a lot more time to spot other optimization opportunities and take advantage of them. A great new technology from Microsoft, called .NET Native, enables you to compile managed code into self-contained executables optimized using the Visual C++ back end. Currently, this technology supports only Windows Store apps.

The ability to control managed code optimizations is currently limited. The C# and Visual Basic compilers only provide the ability to turn on or off optimizations using the /optimize switch. To control JIT optimizations, you can apply the System.Runtime.CompilerServices.MethodImpl attribute on a method with an option from MethodImplOptions specified. The NoOptimization option turns off optimizations, the NoInlining option prevents the method from being inlined, and the AggressiveInlining (.NET 4.5) option gives a recommendation (more than just a hint) to the JIT compiler to inline the method.

## Wrapping Up

All of the optimization techniques discussed in this article can significantly improve the performance of your code by a double-digit percentage, and all of them are supported by the Visual C++ compiler. What makes these techniques important is that, when applied, they enable the compiler to perform other optimizations. This is by no means a comprehensive discussion of the compiler optimizations performed by Visual C++. However, I hope it has given you an appreciation of the capabilities of the compiler. Visual C++ can do more, much more, so stay tuned for Part 2. ■

**Hadi Brais** *is a Ph.D. scholar at the Indian Institute of Technology Delhi (IITD), researching compiler optimizations for the next-generation memory technology. He spends most of his time writing code in C/C++/C# and digging deep into the CLR and CRT. He blogs at hadibrais.wordpress.com. Reach him at hadi.b@live.com.*

**Thanks** to the following Microsoft technical expert for reviewing this article: *Jim Hogg*

# Building an Enterprise Search for .NET

Damian Zapart

The emergence of cloud computing in recent years has been a boon for organizations and users alike. Organizations can know their customers like never before, and target them with personalized communications. Users can get to their data from almost anywhere, making it far more accessible and useful. Huge datacenters have been built all around the world to store all of that data. But Big Data leads to big challenges.

The well-known quote by John Naisbitt, "We are drowning in data but starving for information," in his book, "Megatrends: Ten New Directions Transforming Our Lives" (Warner Books, 1982), perfectly describes the current situation in the Big Data market. Companies are able to store petabytes of data, but being able to make sense of that data, and making it searchable, is far more difficult, especially because most data warehouses store data in a non-structured way (NoSQL) across multiple collections inside particular Big Data stores or even in a distributed form between different warehouses. Moreover, there are a variety of data formats, such as JSON documents, Microsoft Office files and so forth. Searching through a single unstructured collection typically isn't a problem, but it's much harder to search all unstructured data across multiple collections to find just a particular small subset of results when the user has no idea where it might be. This is where enterprise search comes into play.

This article discusses:

- Creating a simple, multi-source search solution
- Setting up Elasticsearch as a Windows Service
- Extending Elasticsearch with plug-ins
- Setting up SQL Server
- Making the data sources searchable
- Searching the data

Technologies discussed:

Microsoft .NET Framework, Elasticsearch, JSON, SQL Server 2014

Code download available at:

msdn.microsoft.com/magazine/msdnmag0215

## Enterprise Search

Here's the essential challenge of enterprise search: How can a large organization with a lot of data sources provide internal and external users with the ability to search all public company data sources through one interface? That single interface might be an API, a company Web site, or even a simple textbox with autocomplete functionality implemented under the hood. No matter which interface a company chooses, it must provide the ability to search through its entire data universe, which might include structured and unstructured databases, intranet documents in different formats, other APIs and other kinds of data sources.

Because searching through multiple datasets is quite complex, there are only a few recognized enterprise search solutions—and the bar is high. An enterprise search solution must include the following features:

- Content awareness: Know where particular types of data can be located.
- Real-time indexing: Keep all data indexed.
- Content processing: Make different data sources accessible.

One of the most popular enterprise search solutions is the open source Elasticsearch (elasticsearch.org). This Java-based server built on

top of Apache Lucene (lucene.apache.org) provides scalable, full-text search over multiple data sources, with JSON support and a REST Web interface, as well as high availability, conflict management and real-time analytics. Visit bit.ly/1vzoUrR to see its full feature set.

From a high level, the way Elasticsearch stores data is very simple. The topmost element of the structure within a server is called an index, and multiple indices can exist in the same data store. The index itself is just a container for documents (one or many), and each document is a collection of one or more fields (with no structures defined). Each index can contain data aggregated into units called types, which represent logical groups of data within a particular index.

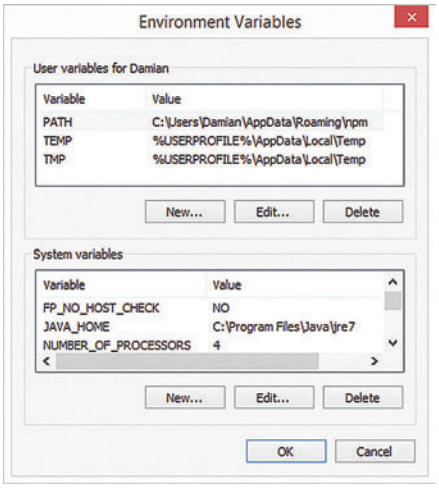



Figure 1 **Setting the Java_Home Environment Variable**

It might be useful to think of Elasticsearch as similar to a table from the world of the relational databases. The same correlation exists between a table's rows and columns and an index's documents and fields, where a document corresponds to a row and a field to a column. However, with Elasticsearch, there's no fixed data structure or database schema.

As I noted, developers can communicate with the Elasticsearch server via a REST Web interface. This means they can query indices, types, data or other system information just by sending REST Web requests from a browser or any other type of Web client. Here are some examples of GET requests:

- Query for all indices:
  `http://localhost:9200/_cat/indices/?v`
- Query for index metadata:
  `http://localhost:9200/clients/_stats`
- Query for all index data:
  `http://localhost:9200/clients/_search?q=*:*`
- Search for a specific field value within the index:
  `http://localhost:9200/clients/_search?q=field:value`
- Getting all data within the index mapping type:
  `http://localhost:9200/clients/orders/_search?q=*:*`

## Creating a Search

To demonstrate how to create a simple, multi-source solution, I'm going to use Elasticsearch 1.3.4 with JSON documents, PDF documents and a SQL Server database. To start, I'll briefly describe Elasticsearch setup and then demonstrate how to plug in each data source to make the data searchable. To keep things simple, I'll present a close-to-real-life example that uses data sources from the well-known Contoso company.

I'll use a SQL Server 2014 database with multiple tables in it, though I'll only be using one, dbo.Orders. As the table name suggests, it stores records about the company's client orders—a huge number of records, yet easy to manage:

```
CREATE TABLE [dbo].[Orders]
(
  [Id] [int] IDENTITY(1,1) NOT NULL primary key,
  [Date] [datetime] NOT NULL,
  [ProductName] [nvarchar](100) NOT NULL,
  [Amount] [int] NOT NULL,
  [UnitPrice] [money] NOT NULL
);
```

I also have a network share with multiple company documents organized in a folder hierarchy. The documents are related to different product marketing campaigns the company organized in the past and are stored in multiple formats, including PDF and Microsoft Office Word. The average document size is approximately 1MB.

Finally, I have an internal company API that exposes the company's client information in JSON format; because I know the structure of the response, I'm able to easily deserialize it to an object of type client. My goal is to make all of the data sources searchable using the Elasticsearch engine. Moreover, I want to create a Web API 2-based Web service that, under the hood, will perform an enterprise query across all indices by making a single call to the Elasticsearch server. (Find more information on Web API 2 at bit.ly/1ae6uya.) The Web service will return the results as a list of suggestions with potential hints to the end user; such a list can later be consumed by an autocomplete control embedded in the ASP.NET MVC application, or by any other kind of Web site.

## Setting Up

The first thing I need to do is install the Elasticsearch server. For Windows, you can do this either automatically or manually, with the same result—a running Windows service hosting the Elasticsearch server. The automatic installation is very quick and easy; all you need to do is to download and run the Elasticsearch MSI installer (bit.ly/12RkHDz). Unfortunately, there's no way to choose a Java version or, much more important, an Elasticsearch version. The manual installation process, in contrast, requires a little more effort, but it allows much more control over the components so it's more suitable in this case.

> Because searching through multiple datasets is quite complex, there are only a few recognized enterprise search solutions—and the bar is high.

Setting up Elasticsearch as a Windows service manually requires the following steps:

1. Download and install the most recent Java SE Runtime Environment (bit.ly/1m1oKlp).
2. Add the environment variable called JAVA_HOME. Its value will be the folder path you've installed Java into (for example, C:\Program Files\Java\jre7), as shown in **Figure 1**.

```
{
   "cluster_name": "elasticsearch",
   "nodes": {
     "1yHhgp6sT72fQ7YxmOn0Rw": {
       "name": "Washout",
       "transport_address": "inet[/169.254.80.80:9300]",
       "host": "Lenovo",
       "ip": "169.254.80.80",
       "version": "1.3.4",
       "build": "a70f3cc",
       "http_address": "inet[/169.254.80.80:9200]",
       "plugins": [
         {
           "name": "jdbc-1.3.4.4-d2e33c3",
           "version": "1.3.4.4",
           "description": "JDBC plugin",
           "jvm": true,
           "site": false
         },
         {
           "name": "mapper-attachments",
           "version": "2.3.2",
           "description": "Adds the attachment type allowing to parse difference attachment formats",
           "jvm": true,
           "site": false
         }
       ]
     }
   }
}
```

Figure 2 **The Installed Plug-Ins**

3. Download the Elasticsearch file (bit.ly/1upadla) and unzip it.
4. Move unzipped sources to Program Files | Elasticsearch (optional).
5. Run the command prompt as an administrator and execute service.bat with the install parameter:

```
C:\Program Files\Elasticsearch\elasticsearch-1.3.4\bin>service.bat install
```

My first goal is to provide full-text search support for attachments.

That's all it takes to get the Windows service up and running, with the Elasticsearch server accessible on localhost, on the port 9200. Now I can make a Web request to the URL http://localhost:9200/ via any Web browser and I'll get a response that looks like this:

```
{
  "status" : 200,
  "name" : "Washout",
  "version" : {
    "number" : "1.3.4",
    "build_hash" : "a70f3ccb52200f8f2c87e9c370c6597448eb3e45",
    "build_timestamp" : "2014-09-30T09:07:17Z",
    "build_snapshot" : false,
    "lucene_version" : "4.9"
  },
  "tagline" : "You Know, for Search"
}
```

Now my local instance of Elasticsearch is ready to go; however, the raw version doesn't give me the ability to connect to SQL Server or run a full-text search through data files. To make these features available, I must also install several plug-ins.

## Extending Elasticsearch

As I mentioned, the raw version of Elasticsearch doesn't let you index an external data source like SQL Server, Office or even a PDF. To make all these data sources searchable I need to install a few plug-ins, which is quite easy.

My first goal is to provide full-text search support for attachments. By attachment, I mean a base64-encoded representation of the source file that has been uploaded to an Elasticsearch data store as a JSON document. (See bit.ly/12RGmvg for information on the attachment type.) The plug-in I need for this purpose is the Mapper Attachments Type for Elasticsearch version 2.3.2, available at bit.ly/1Alj8sy. This is the Elasticsearch extension that enables a full-text search for documents, and it's based on the Apache Tika project (tika.apache.org), which detects and extracts metadata and text content from various types of documents and provides support for the file formats listed at bit.ly/1qEyVmr.

As with most plug-ins for Elasticsearch, this installation is extremely straightforward and all I need to do is run the command prompt as an Administrator and execute the following command:

```
bin>plugin --install elasticsearch/elasticsearch-mapper-attachments/2.3.2
```

After downloading and extracting the plug-in, I need to restart the Elasticsearch Windows service.

When that's done, I need to configure SQL Server support. Of course, there's also a plug-in for this. It's called JDBC River (bit.ly/12CK8Zu) and allows fetching data from a JDBC source, such as SQL Server, for indexing into Elasticsearch. The plug-in is easily installed and configured, though the installation process has three stages that need to be completed.

1. First, I install the Microsoft JDBC Driver 4.0, a Java-based data provider for SQL Server that can be downloaded from

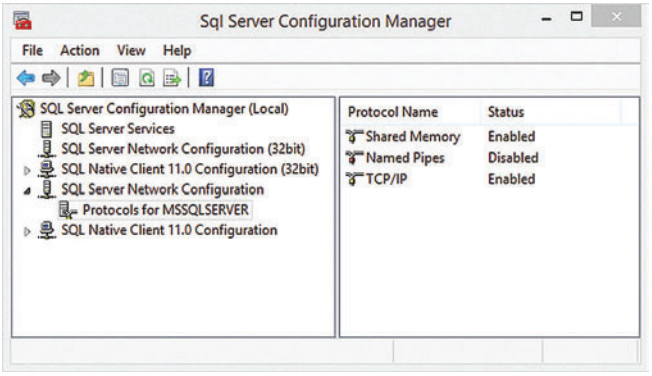



Figure 3 **Enabling TCP/IP for the SQL Server Instance**

bit.ly/1maiM2j. The important thing to remember is I need to extract the content of the downloaded file into the folder called Microsoft JDBC Driver 4.0 for SQL Server (which needs to be created if it doesn't exist), directly under the Program Files folder, so the resulting path looks like: C:\Program Files\Microsoft JDBC Driver 4.0 for SQL Server.

2. Next, I install the plug-in using the following command:

```
bin> plugin --install
jdbc --url "http://xbib.org/repository/org/xbib/elasticsearch/
plugin/elasticsearch-river-jdbc/1.3.4.4/
elasticsearch-river-jdbc-1.3.4.4-plugin.zip"
```

3. Finally, I copy the SQLJDBC4.jar file extracted in the first step (C:\Program Files\Microsoft JDBC DRIVER 4.0 for SQL Server\sqljdbc_4.0\enu\SQLJDBC4.jar) into the lib folder in the Elasticsearch directory (C:\Program Files\Elasticsearch\lib). When I'm done, I need to remember to restart the Windows service.

Now all of the required plug-ins have been installed. However, to verify the installations finished correctly, I need to send the following command as an HTTP GET request:

```
http://localhost:9200/_nodes/_all/plugins
```

In the response, I expect to see both installed plug-ins listed, as shown in **Figure 2**.

## Setting up SQL Server

To use JDBC River with SQL Server, the SQL Server instance needs to be accessible via TCP/IP, which, by default, is disabled. However, enabling it is simple, and all I need to do is open SQL Server Configuration Manager and, under the SQL Server Network Configuration, for the SQL Server instance I want to connect to, change the Status value to Enable for TCP/IP protocol, as shown in **Figure 3**. After doing this, I should be able to log into my SQL Server instance via Management Studio by using localhost 1433 in the Server Name (port 1433 is the default port for accessing SQL Server via TCP/IP).

## Making Sources Searchable

All required plug-ins have been installed, so now it's time to load the data. As mentioned earlier, I have three different data sources (JSON documents, files and an order table from a SQL Server database) that I wish to have indexed on Elasticsearch. I can index these data sources in many different ways, but I want to demonstrate how easy it is using a .NET-based application I've implemented. Therefore, as a pre-condition for indexing, I need to install an external library called NEST (bit.ly/1vZjtCf) for my project, which is nothing but a managed wrapper around an Elasticsearch Web interface. Because this library is available on NuGet, making it part of my project is as simple as executing a single command in the Package Manager Console:

```
PM> Install-Package NEST
```

Now, with the NEST library available in my solution, I can create a new class library project called ElasticSearchRepository. I used this name because I decided to keep all function calls from the ElasticClient class (which is part of the NEST library) separate from the rest of the solution. By doing this, the project becomes similar to the repository design pattern widely applied in Entity Framework-based applications, so it should be easy to understand. Also, in this project I have just three classes: the BaseRepository class, which initializes and exposes the inherited classes and the instance of the ElasticClient, and two other repository classes:

- IndexRepository—a read/write class I'll use to manipulate indices, set mappings and upload documents.
- DiscoveryRepository—a read-only class I'll use during API-based search operations.

**Figure 4** shows the structure of the BaseRepository class with a protected property of ElasticClient type. This type, provided as part of the NEST library, centralizes communication between the Elasticsearch server and client application. To create an instance of it, I can pass a URL to the Elasticsearch server, which I pass as an optional class constructor parameter. If the parameter is null, a default value of http://localhost:9200 will be used.

With the client ready, first I'll index the Client data; this is the easiest scenario because no additional plug-ins are required:

```
public class Client
{
  public int Id { get; set; }

  public string Name { get; set; }

  public string Surname { get; set; }

  public string Email { get; set; }
}
```

To index this type of data, I call the instance of my Elasticsearch client, as well as the Index<T> function, where T is the type of my Client class, which represents serialized data returned from the API. This generic function takes three parameters: the instance of T object class, a target index name and a mapping name within the index:

```
public bool IndexData<T>(T data, string indexName =
  null, string mappingType = null)
  where T : class, new()
  {
    if (client == null)
    {
      throw new ArgumentNullException("data");
    }

    var result = this.client.Index<T>(data,
      c => c.Index(indexName).Type(mappingType));
    return result.IsValid;
  }
```

The last two parameters are optional because NEST will apply its default logic for creating a target index name based on the generic type.

Figure 4 **The BaseRepository Class**

```
namespace ElasticSearchRepository
{
  using System;
  using Nest;

  public class BaseRepository
  {
    protected ElasticClient client;

    public BaseRepository(Uri elastiSearchServerUrl = null)
    {
      this.client = elastiSearchServerUrl != null ?
        new ElasticClient(new ConnectionSettings(elastiSearchServerUrl)) :
        : new ElasticClient();
    }
  }
}
```

Figure 5 **HTTP PUT Request to Create a New JDBC River Mapping**

```
PUT http://localhost:9200/_river/orders_river/_meta
{
"type":"jdbc",
"jdbc":
  {
  "driver": "com.microsoft.sqlserver.jdbc.SQLServerDriver",
  "url":"jdbc:sqlserver://127.0.0.1:1433;databaseName=Contoso",
  "user":"elastic",
  "password":"asd",
  "sql":"SELECT *  FROM dbo.Orders",
  "index" : "clients",
  "type" : "orders",
  "schedule": "0/30 0-59 0-23 ? * *"
  }
}
```

Now I want to index the marketing documents, which are related to the company products. As I have these files stored in a network share, I can wrap information about each particular document into a simple MarketingDocument class. It's worth noting here that if I want a document to be indexed in Elasticsearch, I need to upload it as a Base64-encoded string:

```
public class MarketingDocument
{
  public int Id { get; set; }
  public string Title { get; set; }
  public string ProductName { get; set; }

  // Base64-encoded file content.
  public string Document { get; set; }
}
```

> In theory, I can create a JDBC River mapping by using a Raw function to send a raw request with JSON, but I don't want to overcomplicate things.

The class is ready, so I can use the ElasticClient to mark a particular field in my MarketingDocument class as an attachment. I can achieve this by creating a new index called "products" and adding a new marketing mapping to it (for simplicity, the class name will be the mapping name):

```
private void CreateMarketingIndex()
  {
    client.CreateIndex("products", c =>
      c.AddMapping<Marketing>
      (m => m.Properties(ps =>ps.Attachment(a =>
            a.Name(o =>o.Document).TitleField(t =>
            t.Name(x => x.Name)
            TermVector(TermVectorOption.WithPositionsOffsets)
        )))));
  }
```

Now that I have both the .NET type and the mapping for the marketing data, as well as the definition for an attachment, I can start indexing my files in the same way I indexed the client data:

```
var documents = GetMarketingDocumentsMock();
documents.ForEach((document) =>
{
  indexRepository.IndexData<MarketingDocument>(document, "marketing");
});
```

The final step is the JDBC River setup on Elasticsearch. Unfortunately, NEST doesn't support JDBC River yet. In theory, I can create a JDBC River mapping by using a Raw function to send a raw request with JSON, but I don't want to overcomplicate things. Therefore, to complete the mapping creation process, I'm going to specify the following parameters:

- A connection string to SQL Server database
- A SQL query, which will be used to query for data
- Update schedule
- Target index name and type (optional)

(You'll find a full list of configurable parameters at bit.ly/12CK8Zu.)

To create a new JDBC River mapping, I need to send a PUT request with a request body specified in it to the following URL:

```
http://localhost:9200/_river/{river_name}/_meta
```

In the example in **Figure 5**, I put a request body to create a new JDBC River mapping, which connects to the Contoso database hosted on the local SQL Server instance, which is accessible on port 1433 via TCP/IP.

It uses the login "elastic" and the password "asd" to authenticate the user and execute the following SQL command:

```
SELECT * FROM dbo.Orders
```

Each row of data returned by this SQL query will be indexed under the clients index in the orders mapping type and the indexing will take place every 30 seconds (represented in the Cron notation, see bit.ly/1hCcmnN for more information).

When this process completes, you should see in the Elasticsearch log file (/logs/ elasticsearch.log) information similar to the following:

Figure 6 **Implementation of Two Types of Search**

```
namespace ElasticSearchRepository
{
  using System;
  using System.Collections.Generic;
  using System.Linq;

  public class DiscoveryRepository : BaseRepository
  {

    public DiscoveryRepository(Uri elastiSearchServerUrl = null)
      : base(elastiSearchServerUrl)
    {
    }
    ///<summary>
    public List<Tuple<string, string>> SearchAll(string queryTerm)
    {
      var queryResult=this.client.Search<dynamic>(d =>
        d.AllIndices()
        .AllTypes()
        .QueryString(queryTerm));
      return queryResult
        .Hits
        .Select(c => new Tuple<string, string>(
          c.Indexc.Source.Name.Value))
        .Distinct()
        .ToList();
    }
    ///<summary>
    public dynamic FuzzySearch(string queryTerm)
    {
      return this.client.Search<dynamic>(d =>
        d.AllIndices()
        .AllTypes()
        .Query(q => q.Fuzzy(f =>
          f.Value(queryTerm))));
    }
  }
}
```


Figure 7 **API Search Results for the Term "scrum"**

```
[2014-10-2418:39:52,190][INFO][river.jdbc.RiverMetrics]
pipeline org.xbib.elasticsearch.plugin.jdbc.RiverPipeline@70f0a80d
complete: river jdbc/orders_river metrics: 34553 rows, 6.229481683638776
mean, (0.0 0.0 0.0), ingest metrics: elapsed 2 seconds, 364432.0 bytes
bytes, 1438.0 bytes avg, 0.1 MB/s
```

If something is wrong with the river configuration, the error message will also be in the log.

## Searching the Data

Once all the data has been indexed in the Elasticsearch engine, I can start querying. Of course, I can send simple requests to the Elasticsearch server to query one or multiple indices and mapping types at the same time, but I want to build something more useful and closer to a real-life scenario. So I'm going to split my project into three different components. The first component, which I already presented, is Elasticsearch, which is available via http://localhost:9200/. The second component is an API I'm going to build using Web API 2 technology. The last component is a console application I'll use to set up my indices on Elasticsearch, as well as to feed it with data.

To create my new Web API 2 project, first I need to create an empty ASP.NET Web Application project and then, from the Package Manager Console, run the following install command:

```
Install-Package Microsoft.AspNet.WebApi
```

After the project is created, the next step is to add a new controller, which I'll use to process query requests from the client and pass them to Elasticsearch. Adding a new controller named DiscoveryController involves nothing more than adding a new item, a Web API ApiController class (v2.1). And I need to implement a Search function, which will be exposed via the URL: http://website/api/discovery/search?searchTerm=user_input:

```
[RoutePrefix("api/discovery")]
public class DiscoveryController : ApiController
{
  [HttpGet]
  [ActionName("search")]
  public IHttpActionResult Search(string searchTerm)
  {
    var discoveryRepository = new DiscoveryRepository();
    var result = discoveryRepository.Search(searchTerm);
    return this.Ok(result);
  }
}
```

If the Web API 2 engine can't serialize a response because of a self-referencing loop, you'll have to add the following in the WebApiConfig.cs file, which is located in the AppStart folder:

```
GlobalConfiguration.Configuration
.Formatters
.JsonFormatter
.SerializerSettings
.ReferenceLoopHandling =
  ReferenceLoopHandling.Ignore;
```

As **Figure 6** shows, in the body of the controller I created, I instantiated a class of DiscoveryRepository type, which is just a wrapper around the ElasticClient type from the NEST library. Inside this non-generic, read-only repository I implemented two types of search functions and both of them return a dynamic type. This part is important because by doing this in both function bodies, I'm not limiting my queries just to one index; instead, I'm querying all indices and all types at the same time. This means my results will have a different structure (will be of different types). The only difference between the functions is the query method. In the first function I just use a QueryString method (bit.ly/1mQEEg7), which is an exact match search, and in the second one, a Fuzzy method (bit.ly/1uCk7Ba), which performs a fuzzy search across indices.

Now when my API is ready, I can run and start testing it just by sending GET requests to http://website:port/api/discovery/search?searchTerm=user_input, and pass user input as the value of the searchTerm query parameter. Therefore, **Figure 7** shows the results that my API generates for the search term "scrum." As I highlighted on the screenshot, a search function performed a query over all indices in the data stores and returned hits from multiple indices at the same time.

By implementing the API layer I created the possibility of implementing multiple clients (like a Web site or mobile app), which will be able to consume it. This provides the ability of giving enterprise search functionality to end users. You can find an example implementation of the autocomplete control for an ASP.NET MVC 4-based Web client on my blog at bit.ly/1yThHiZ.

## Wrapping Up

Big Data has brought a lot to the technology marketplace, in terms of both opportunities and challenges. One of the challenges, which is also potentially a great opportunity, is the possibility of implementing fast search across petabytes of data without having to know the exact location of the data in the data universe. In this article I described how to implement enterprise search and demonstrated how to do it in the .NET Framework in combination with the Elasticsearch and NEST libraries. ■

**Damian Zapart** *is a development lead at Citigroup Inc. and focuses mainly on enterprise solutions. At the same time he's a programming geek interested in cutting-edge technologies, design patterns and Big Data. Visit his blog at bit.ly/1suoKN0 to read more about him.*

**Thanks** to the following technical experts for reviewing this article: *Evren Onem (D&B) and Bruno Terkaly (Microsoft)*

# NAVIGATE THE .NET HIGHWAY

## DEVELOPMENT TRACKS INCLUDE:

- Visual Studio/.NET
- JavaScript/HTML5 Client
- ASP.NET
- Mobile Client
- Windows Client
- Database and Analytics
- Cloud Computing
- SharePoint/Office

## Register NOW and Save $300!

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

**Use promo code VAUFEB2**

**vslive.com/austin**

## Join us on the Ultimate Code Trip in 2015!

| LAS VEGAS Code Trip | AUSTIN Code Trip | SAN FRANCISCO Code Trip | REDMOND Code Trip | NEW YORK Code Trip | ORLANDO Code Trip |
|---|---|---|---|---|---|
| March 16-20 | June 1-4 | June 15-18 | August 10-14 | Sept. 28- Oct. 1 | Nov. 16-20 |

**FOLLOW US**

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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



JAMES MCCAFFREY

# L1 and L2 Regularization for Machine Learning

L1 regularization and L2 regularization are two closely related techniques that can be used by machine learning (ML) training algorithms to reduce model overfitting. Eliminating overfitting leads to a model that makes better predictions. In this article I'll explain what regularization is from a software developer's point of view. The ideas behind regularization are a bit tricky to explain, not because they're difficult, but rather because there are several interrelated ideas

In this article I illustrate regularization with logistic regression (LR) classification, but regularization can be used with many types of machine learning, notably neural network classification. The goal of LR classification is to create a model that predicts a variable that can take one of two possible values. For example, you might want to predict the result for a football team (lose = 0, win = 1) in an upcoming game based on the team's current winning percentage (x1), field location (x2), and number of players absent due to injury (x3).

If Y is the predicted value, an LR model for this problem would take the form:

```
z = b0 + b1(x1) + b2(x2) + b3(x3)
Y = 1.0 / (1.0 + e^-z)
```

Here b0, b1, b2 and b3 are weights, which are just numeric values that must be determined. In words, you compute a value z that is the sum of input values times b-weights, add a b0 constant, then pass the z value to the equation that uses math constant e. It turns out that Y will always be between 0 and 1. If Y is less than 0.5, you conclude the predicted output is 0 and if Y is greater than 0.5 you conclude the output is 1. Notice that if there are n features, there will be n+1 b-weights.

For example, suppose a team currently has a winning percentage of 0.75, and will be playing at their opponent's field (-1), and has 3 players out due to injury. And suppose b0 = 5.0, b1 = 8.0, b2 = 3.0, and b3 = -2.0. Then z = 5.0 + (8.0)(0.75) + (3.0)(-1) + (-2.0)(3) = 2.0 and so Y = 1.0 / (1.0 + e^-2.0) = 0.88. Because Y is greater than 0.5, you'd predict the team will win their upcoming game.

I think the best way to explain regularization is by examining a concrete example. Take a look at the screenshot of a demo program in **Figure 1**. Rather than use real data, the demo program begins by generating 1,000 synthetic data items. Each item has 12 predictor variables (often called "features" in ML terminology). The dependent variable value is in the last column. After creating the 1,000 data items, the data set was randomly split into an 800-item training set to be used to find the model b-weights, and a 200-item test set to be used to evaluate the quality of the resulting model.

Next, the demo program trained the LR classifier, without using regularization. The resulting model had 85.00 percent accuracy on the training data, and 80.50 percent accuracy on the test data. The 80.50 percent accuracy is the more relevant of the two values, and is a rough estimate of how accurate you could expect the model to be when presented with new data. As I'll explain shortly, the model was over-fitted, leading to mediocre prediction accuracy.

Next, the demo did some processing to find a good L1 regularization weight and a good L2 regularization weight. Regularization weights are single numeric values that are used by the regularization process. In the demo, a good L1 weight was determined to be 0.005 and a good L2 weight was 0.001.

> Regularization can be used with many types of machine learning, notably neural network classification.

The demo first performed training using L1 regularization and then again with L2 regularization. With L1 regularization, the resulting LR model had 95.00 percent accuracy on the test data, and with L2 regularization, the LR model had 94.50 percent accuracy on the test data. Both forms of regularization significantly improved prediction accuracy.

This article assumes you have at least intermediate programming skills, but doesn't assume you know anything about L1 or L2 regularization. The demo program is coded using C#, but you shouldn't have too much difficulty refactoring the code to another language such as JavaScript or Python.

The demo code is too long to present here, but complete source code is available in the code download that accompanies this article. The demo code has all normal error checking removed to keep the main ideas as clear as possible and the size of the code small.

## Overall Program Structure

The overall program structure, with some minor edits to save space, is presented in **Figure 2**. To create the demo, I launched Visual Studio

Code download available at msdn.microsoft.com/magazine/msdnmag0215.

and created a new C# console application named Regularization. The demo has no significant Microsoft .NET Framework dependencies, so any recent version of Visual Studio will work.

After the template code loaded into the Visual Studio editor, in the Solution Explorer window I renamed file Program.cs to the more descriptive RegularizationProgram.cs and Visual Studio automatically renamed class Program for me. At the top of the source code, I deleted all using statements that pointed to unneeded namespaces, leaving just the reference to the top-level System namespace.

All of the logistic regression logic is contained in a single LogisticClassifier class. The LogisticClassifier class contains a nested helper Particle class to encapsulate particle swarm optimization (PSO), the optimization algorithm used for training. Note that the LogisticClassifier class contains a method Error, which accepts parameters named alpha1 and alpha2. These parameters are the regularization weights for L1 and L2 regularization.

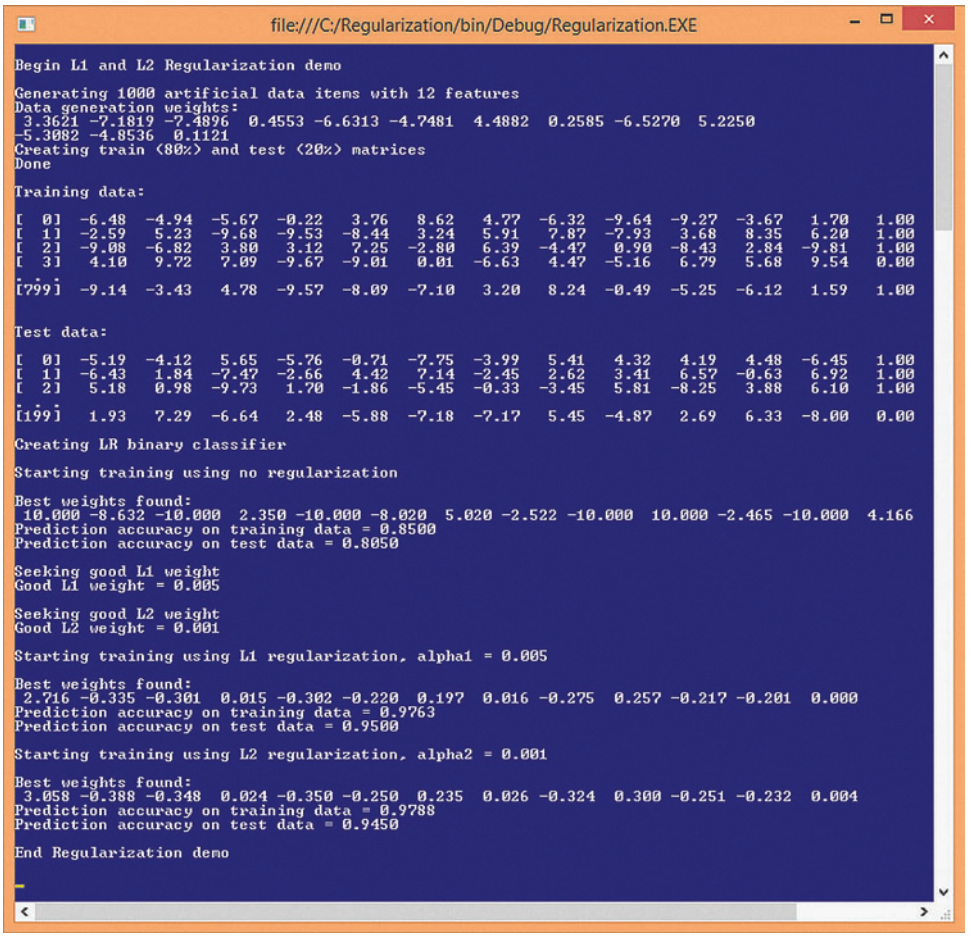



Figure 1 **Regularization with Logistic Regression Classification**

In the Main method, the synthetic data is created with these statements:

```
int numFeatures = 12;
int numRows = 1000;
int seed = 42;
double[][] allData = MakeAllData(numFeatures, numRows, seed);
```

The seed value of 42 was used only because that value gave nice, representative demo output. Method MakeAllData generates 13 random weights between -10.0 and +10.0 (one weight for each feature, plus the b0 weight). Then the method iterates 1,000 times. On each iteration, a random set of 12 input values is generated, then an intermediate logistic regression output value is calculated using the random weights. An additional random value is added to the output to make the data noisy and more prone to overfitting.

The data is split into an 800-item set for training and a 200-item set for model evaluation with these statements:

```
double[][] trainData;
double[][] testData;
MakeTrainTest(allData, 0, out trainData, out testData);
```

A logistic regression prediction model is created with these statements:

```
LogisticClassifier lc = new LogisticClassifier(numFeatures);
int maxEpochs = 1000;
double[] weights = lc.Train(trainData, maxEpochs, seed, 0.0, 0.0);
ShowVector(weights, 4, weights.Length, true);
```

Variable maxEpochs is a loop counter limiting value for the PSO training algorithm. The two 0.0 arguments passed to method Train are the L1 and L2 regularization weights. By setting those weights to 0.0, no regularization is used. The model's quality is evaluated with these two statements:

```
double trainAccuracy = lc.Accuracy(trainData, weights);
double testAccuracy = lc.Accuracy(testData, weights);
```

One of the downsides to using regularization is that the regularization weights must be determined. One approach for finding good regularization weights is to use manual trial and error, but a programmatic technique is usually better. A good L1 regularization weight is found and then used with these statements:

```
double alpha1 = lc.FindGoodL1Weight(trainData, seed);
weights = lc.Train(trainData, maxEpochs, seed, alpha1, 0.0);
trainAccuracy = lc.Accuracy(trainData, weights);
testAccuracy = lc.Accuracy(testData, weights);
```

The statements for training the LR classifier using L2 regularization are just like those for using L1 regularization:

```
double alpha2 = lc.FindGoodL2Weight(trainData, seed);
weights = lc.Train(trainData, maxEpochs, seed, 0.0, alpha2);
trainAccuracy = lc.Accuracy(trainData, weights);
testAccuracy = lc.Accuracy(testData, weights);
```

In the demo, the alpha1 and alpha2 values were determined using the LR object public-scope methods FindGoodL1Weight and FindGoodL2Weight and then passed to method Train. An alternative design is suggested by calling this code:

```
bool useL1 = true;
bool useL2 = false:
lc.Train(traiData, maxEpochs, useL1, useL2);
```

This design approach allows the training method to determine the regularization weights and leads to a bit cleaner interface.

## Understanding Regularization

Because L1 and L2 regularization are techniques to reduce model overfitting, in order to understand regularization, you must understand overfitting. Loosely speaking, if you train a model too much, you will eventually get weights that fit the training data extremely well, but when you apply the resulting model to new data, the prediction accuracy is very poor.

Overfitting is illustrated by the two graphs in **Figure 3**. The first graph shows a hypothetical situation where the goal is to classify two types of items, indicated by red and green dots. The smooth blue curve represents the true separation of the two classes, with red dots belonging above the curve and green dots belonging below the curve. Notice that because of random errors in the data, two of the red dots are below the curve and two green dots are above the curve. Good training, where overfitting doesn't occur, would result in weights that correspond to the smooth blue curve. Suppose a new data point came in at (3, 7). The data item would be above the curve and be correctly predicted to be class red.

Figure 2 **Overall Program Structure**

```
using System;
namespace Regularization
{
  class RegularizationProgram
  {
    static void Main(string[] args)
    {
      Console.WriteLine("Begin L1 and L2 Regularization demo");

      int numFeatures = 12;
      int numRows = 1000;
      int seed = 42;

      Console.WriteLine("Generating " + numRows +
        " artificial data items with " + numFeatures + " features");
      double[][] allData = MakeAllData(numFeatures, numRows, seed);

      Console.WriteLine("Creating train and test matrices");
      double[][] trainData;
      double[][] testData;
      MakeTrainTest(allData, 0, out trainData, out testData);

      Console.WriteLine("Training data: ");
      ShowData(trainData, 4, 2, true);

      Console.WriteLine("Test data: ");
      ShowData(testData, 3, 2, true);

      Console.WriteLine("Creating LR binary classifier");
      LogisticClassifier lc = new LogisticClassifier(numFeatures);

      int maxEpochs = 1000;
      Console.WriteLine("Starting training using no regularization");
      double[] weights = lc.Train(trainData, maxEpochs,
        seed, 0.0, 0.0);

      Console.WriteLine("Best weights found:");
      ShowVector(weights, 3, weights.Length, true);

      double trainAccuracy = lc.Accuracy(trainData, weights);
      Console.WriteLine("Prediction accuracy on training data = " +
        trainAccuracy.ToString("F4"));

      double testAccuracy = lc.Accuracy(testData, weights);
      Console.WriteLine("Prediction accuracy on test data = " +
        testAccuracy.ToString("F4"));

      Console.WriteLine("Seeking good L1 weight");
      double alpha1 = lc.FindGoodL1Weight(trainData, seed);
      Console.WriteLine("L1 weight = " + alpha1.ToString("F3"));

      Console.WriteLine("Seeking good L2 weight");
      double alpha2 = lc.FindGoodL2Weight(trainData, seed);
      Console.WriteLine("L2 weight = " + alpha2.ToString("F3"));

      Console.WriteLine("Training with L1 regularization, " +
        "alpha1 = " + alpha1.ToString("F3"));
      weights = lc.Train(trainData, maxEpochs, seed, alpha1, 0.0);

      Console.WriteLine("Best weights found:");
      ShowVector(weights, 3, weights.Length, true);

      trainAccuracy = lc.Accuracy(trainData, weights);
      Console.WriteLine("Prediction accuracy on training data = " +
        trainAccuracy.ToString("F4"));

      testAccuracy = lc.Accuracy(testData, weights);
      Console.WriteLine("Prediction accuracy on test data = " +
        testAccuracy.ToString("F4"));

      Console.WriteLine("Training with L2 regularization, " +
        "alpha2 = " + alpha2.ToString("F3"));
      weights = lc.Train(trainData, maxEpochs, seed, 0.0, alpha2);

      Console.WriteLine("Best weights found:");
      ShowVector(weights, 3, weights.Length, true);

      trainAccuracy = lc.Accuracy(trainData, weights);
      Console.WriteLine("Prediction accuracy on training data = " +
        trainAccuracy.ToString("F4"));

      testAccuracy = lc.Accuracy(testData, weights);
      Console.WriteLine("Prediction accuracy on test data = " +
        testAccuracy.ToString("F4"));

      Console.WriteLine("End Regularization demo");
      Console.ReadLine();
    }

    static double[][] MakeAllData(int numFeatures,
      int numRows, int seed) { . . }

    static void MakeTrainTest(double[][] allData, int seed,
      out double[][] trainData, out double[][] testData) { . . }

    public static void ShowData(double[][] data, int numRows,
      int decimals, bool indices) { . . }

    public static void ShowVector(double[] vector, int decimals,
      int lineLen, bool newLine) { . . }
  }

  public class LogisticClassifier
  {
    private int numFeatures;
    private double[] weights;
    private Random rnd;

    public LogisticClassifier(int numFeatures) { . . }
    public double FindGoodL1Weight(double[][] trainData,
      int seed) { . . }
    public double FindGoodL2Weight(double[][] trainData,
      int seed) { . . }
    public double[] Train(double[][] trainData, int maxEpochs,
      int seed, double alpha1, double alpha2) { . . }
    private void Shuffle(int[] sequence) { . . }
    public double Error(double[][] trainData, double[] weights,
      double alpha1, double alpha2) { . . }
    public double ComputeOutput(double[] dataItem,


      double[] weights) { . . }
    public int ComputeDependent(double[] dataItem,
      double[] weights) { . . }
    public double Accuracy(double[][] trainData,
      double[] weights) { . . }

    public class Particle { . . }
  }
} // ns
```

# We know how application development can be.

## Let DotImage take some of the bite out of your challenges.

Atalasoft DotImage

Connecting the dots is a no-brainer. DotImage image-enables your .NET-based web application faster, more cost effectively, and less painfully than if done on your own. This proven SDK is versatile, with options including OCR capabilities, WingScan compatibility, and support for a range of formats. Coupled with dedicated assistance from our highly knowledgeable and skilled engineers, DotImage helps your business connect with powerful information hidden inside your documents, making the big picture much easier to see.

Image-enabling experts & bacon connoisseurs. Visit us online to see our full line of SDK products for .NET and Java

Atalasoft
A Kofax Company

www.atalasoft.com

The second graph in **Figure 3** has the same dots but a different blue curve that is a result of overfitting. This time all the red dots are above the curve and all the green dots are below the curve. But the curve is too complex. A new data item at (3, 7) would be below the curve and be incorrectly predicted as class green.

Overfitting generates non-smooth prediction curves, in other words, those that are not "regular." Such poor, complex prediction curves are usually characterized by weights that have very large or very small values. Therefore, one way to reduce overfitting is to prevent model weights from becoming very small or large. This is the motivation for regularization.

When an ML model is being trained, you must use some measure of error to determine good weights. There are several different ways to measure error. One of the most common techniques is the mean squared error, where you find the sum of squared differences between the computed output values for a set of weight values and the known, correct output values in the training data, and then divide that sum by the number of training items. For example, suppose for a candidate set of logistic regression weights, with just three training items, the computed outputs and correct output values (sometimes called the desired or target values) are:

```
computed  desired
  0.60      1.0
  0.30      0.0
  0.80      1.0
```

Here, the mean squared error would be:

((0.6 - 1.0)^2 + (0.3 - 0.0)^2 + (0.8 - 1.0)^2) / 3 =
(0.16 + 0.09 + 0.04) / 3 =
0.097

Expressed symbolically, mean squared error can be written:

E = Sum(o - t)^2 / n

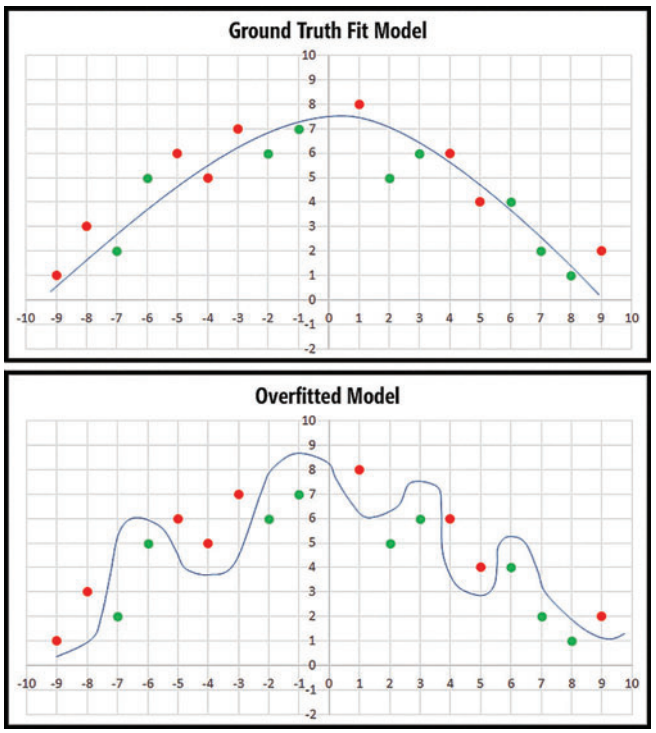



Figure 3 **Model Overfitting**

where Sum represents the accumulated sum over all training items, o represents computed output, t is target output and n is the number of training data items. The error is what training minimizes using one of about a dozen numerical techniques with names like gradient descent, iterative Newton-Raphson, L-BFGS, back-propagation and swarm optimization.

In order to prevent the magnitude of model weight values from becoming large, the idea of regularization is to penalize weight values by adding those weight values to the calculation of the error term. If weight values are included in the total error term that's being minimized, then smaller weight values will generate smaller error values. L1 weight regularization penalizes weight values by adding the sum of their absolute values to the error term. Symbolically:

E = Sum(o - t)^2 / n + Sum(Abs(w))

> Therefore, one way to reduce overfitting is to prevent model weights from becoming very small or large.

L2 weight regularization penalizes weight values by adding the sum of their squared values to the error term. Symbolically:

E = Sum(o - t)^2 / n + Sum(w^2)

Suppose for this example there are four weights to be determined and their current values are (2.0, -3.0, 1.0, -4.0). The L1 weight penalty added to the 0.097 mean squared error would be (2.0 + 3.0 + 1.0 + 4.0) = 10.0. The L2 weight penalty would be 2.0^2 + -3.0^2 + 1.0^2 + -4.0^2 = 4.0 + 9.0 + 1.0 + 16.0 = 30.0.

To summarize, large model weights can lead to overfitting, which leads to poor prediction accuracy. Regularization limits the magnitude of model weights by adding a penalty for weights to the model error function. L1 regularization uses the sum of the absolute values of the weights. L2 regularization uses the sum of the squared values of the weights.

## Why Two Different Kinds of Regularization?

L1 and L2 regularization are similar. Which is better? The bottom line is that even though there are some theory guidelines about which form of regularization is better in certain problem scenarios, in my opinion, in practice you must experiment to find which type of regularization is better, or whether using regularization at all is better.

As it turns out, using L1 regularization can sometimes have a beneficial side effect of driving one or more weight values to 0.0, which effectively means the associated feature isn't needed. This is one form of what's called feature selection. For example, in the demo run in **Figure 1**, with L1 regularization the last model weight is 0.0. This means the last predictor value doesn't contribute to the LR model. L2 regularization limits model weight values, but usually doesn't prune any weights entirely by setting them to 0.0.

So, it would seem that L1 regularization is better than L2 regularization. However, a downside of using L1 regularization is that the

technique can't be easily used with some ML training algorithms, in particular those algorithms that use calculus to compute what's called a gradient. L2 regularization can be used with any type of training algorithm.

> As it turns out, using L1 regularization can sometimes have a beneficial side effect of driving one or more weight values to 0.0, which effectively means the associated feature is not needed.

To summarize, L1 regularization sometimes has a nice side effect of pruning out unneeded features by setting their associated weights to 0.0 but L1 regularization doesn't easily work with all forms of training. L2 regularization works with all forms of training, but doesn't give you implicit feature selection. In practice, you must use trial and error to determine which form of regularization (or neither) is better for a particular problem.

## Implementing Regularization

Implementing L1 and L2 regularization is relatively easy. The demo program uses PSO training with an explicit error function, so all that's necessary is to add the L1 and L2 weight penalties. The definition of method Error begins with:

```
public double Error(double[][] trainData, double[] weights,
  double alpha1, double alpha2)
{
  int yIndex = trainData[0].Length - 1;
  double sumSquaredError = 0.0;
  for (int i = 0; i < trainData.Length; ++i)
  {
    double computed = ComputeOutput(trainData[i], weights);
    double desired = trainData[i][yIndex];
    sumSquaredError += (computed - desired) * (computed - desired);
  }
...
```

The first step is to compute the mean squared error by summing the squared differences between computed outputs and target outputs. (Another common form of error is called cross-entropy error.) Next, the L1 penalty is calculated:

```
double sumAbsVals = 0.0; // L1 penalty
for (int i = 0; i < weights.Length; ++i)
  sumAbsVals += Math.Abs(weights[i]);
```

Then the L2 penalty is calculated:

```
double sumSquaredVals = 0.0; // L2 penalty
for (int i = 0; i < weights.Length; ++i)
  sumSquaredVals += (weights[i] * weights[i]);
```

Method Error returns the MSE plus the penalties:

```
...
  return (sumSquaredError / trainData.Length) +
    (alpha1 * sumAbsVals) +
    (alpha2 * sumSquaredVals);
}
```

The demo uses an explicit error function. Some training algorithms, such as gradient descent and back-propagation, use the error function implicitly by computing the calculus partial derivative (called the gradient) of the error function. For those training algorithms, to use L2 regularization (because the derivative of w^2 is 2w), you just add a 2w term to the gradient (although the details can be a bit tricky).

## Finding Good Regularization Weights

There are several ways to find a good (but not necessarily optimal) regularization weight. The demo program sets up a set of candidate values, computes the error associated with each candidate, and returns the best candidate found. The method to find a good L1 weight begins:

```
public double FindGoodL1Weight(double[][] trainData, int seed)
{
  double result = 0.0;
  double bestErr = double.MaxValue;
  double currErr = double.MaxValue;
  double[] candidates = new double[] { 0.000, 0.001, 0.005,
    0.010, 0.020, 0.050, 0.100, 0.150 };
  int maxEpochs = 1000;
  LogisticClassifier c =
    new LogisticClassifier(this.numFeatures);
```

Adding additional candidates would give you a better chance of finding an optimal regularization weight at the expense of time. Next, each candidate is evaluated, and the best candidate found is returned:

```
  for (int trial = 0; trial < candidates.Length; ++trial) {
    double alpha1 = candidates[trial];
    double[] wts = c.Train(trainData, maxEpochs, seed, alpha1, 0.0);
    currErr = Error(trainData, wts, 0.0, 0.0);
    if (currErr < bestErr) {
      bestErr = currErr; result = candidates[trial];
    }
  }
  return result;
}
```

Notice the candidate regularization weight is used to train the evaluation classifier, but the error is computed without the regularization weight.

## Wrapping Up

Regularization can be used with any ML classification technique that's based on a mathematical equation. Examples include logistic regression, probit classification and neural networks. Because it reduces the magnitudes of the weight values in a model, regularization is sometimes called weight decay. The major advantage of using regularization is that it often leads to a more accurate model. The major disadvantage is that it introduces an additional parameter value that must be determined, the regularization weight. In the case of logistic regression this isn't too serious because there's usually just the learning rate parameter, but when using more complex classification techniques, neural networks in particular, adding another so-called hyperparameter can create a lot of additional work to tune the combined values of the parameters. ■

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

**Thanks** to the following technical expert at Microsoft Research for reviewing this article: *Richard Hughes*



TED NEWARD AND
JOE HUMMEL, PH.D.

# Rise of Roslyn, Part 2: Writing Diagnostics

By now, readers will have heard much of the buzz surrounding the strategies Microsoft seems to be pursuing for the next generation of Microsoft developer tools: more open source, more cross-platform, more openness and more transparency. "Roslyn"—the code name for the .NET Compiler Platform project—forms a major part of that story, being the first time that Microsoft has really committed production-quality compiler tool infrastructure to an open development model. With the announcement that Roslyn is now the compiler used by the Microsoft .NET Framework teams themselves to build .NET, Roslyn has achieved a certain degree of "inception": The platform and its language tools are now being built by the platform and its language tools. And, as you'll see in this article, you can use the language tools to build more language tools to help you build for the platform.

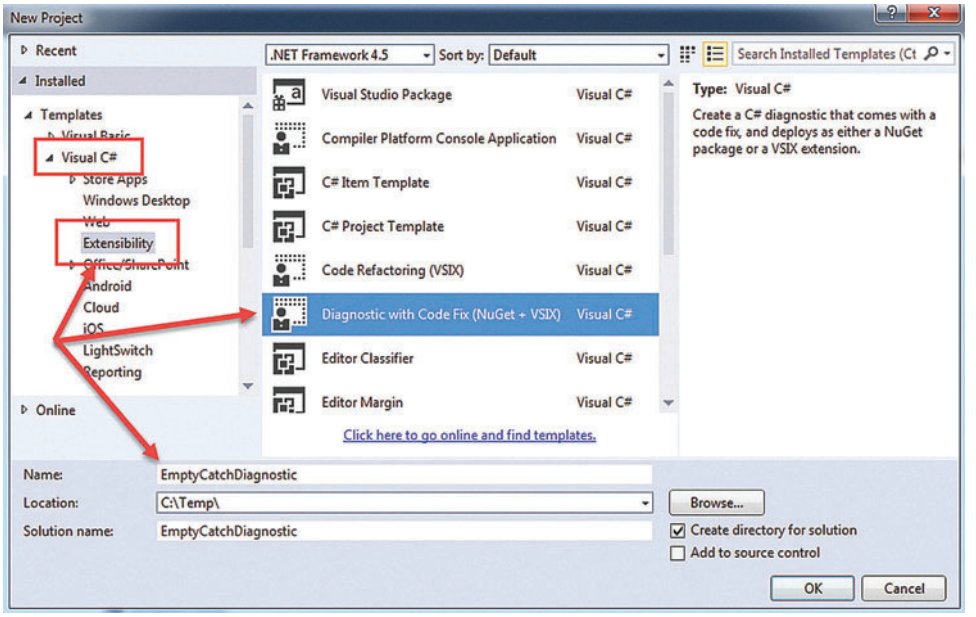



Figure 1 **Diagnostic with Code Fix (NuGet + VSIX) Project Template**

Confused? Don't be—it'll all make sense in just a bit.

## 'But We Don't *Do* That'

Since the first programmer started working with the second programmer—and found him "doing it wrong," at least in the first programmer's opinion—teams have struggled to create some semblance of unity and consistency in the way code is written, the degree of error checking done, the manner in which objects are used and so on. Historically, this has been the province of "coding standards," essentially a set of rules that every programmer is supposed to follow when writing code for the company. Sometimes, programmers even go so far as to read them. But without any sort of coherent and consistent enforcement—usually through that time-honored practice of "code review" during which everybody bickers over where the curly braces should go and what the variables should be named—coding standards really end up having little impact overall on code quality.

Over time, as language tools got more mature, developers started looking to tools themselves to provide this level of enforcement. After all, if there's one thing a computer is good at, it's repeatedly performing the same kinds of detailed analysis, over and over again, without fail or hesitation or mistake. Remember, that's part of the job of a compiler in the first place: Discover common human mistakes that can lead to error-prone code, and fail early so programmers are required to fix them before end users see them. Tools that analyze code, looking for error patterns, are called "static analysis tools" and can help identify bugs long before you even run the unit tests.

Historically in the .NET Framework, it's been difficult to build and maintain such tools. Static analysis tools require a significant development effort and must be updated as languages and libraries evolve; for companies working in both C# and Visual Basic .NET, the effort doubles. Binary analysis tools, such as FxCop, work at the Intermediate Language (IL) level, avoiding language complexities. However, at the very least, there's a structural loss of information in the translation from source to IL, making it that much more difficult to relate issues back to the level where the programmer is working—the

This article discusses prerelease versions of Visual Studio 2015 and the .NET Compiler Platform, code-named "Roslyn." All related information is subject to change.

source. Binary analysis tools also run after compilation, preventing IntelliSense-like feedback during the programming process.

Roslyn, however, was built from the beginning to be extended. Roslyn uses the term "analyzer" to describe source-code analysis extensions that can—and do—run in the background while developers are programming. By creating an analyzer, you can ask Roslyn to enforce additional, higher-order kinds of "rules," helping to eliminate bugs without having to run additional tools.

## What Could Go Wrong?

It's a sad, sad day to admit this, but periodically we see code like this:

```
try
{
  int x = 5; int y = 0;
  // Lots of code here
  int z = x / y;
}
catch (Exception ex)
{
  // TODO: come back and figure out what to do here
}
```

Often, that TODO is written with the best of intentions. But, as the old saying goes, the road to perdition is paved with good intentions. Naturally, the coding standard says this is bad, but it's only a violation if somebody catches you. Sure, a text-file scan would reveal the "TODO," but the code is littered with TODOs, none of which are hiding errors as ugly as this. And, of course, you'll only find this line of code after a major demo bombs silently and you slowly, painfully backtrack the devastation until you find that this code, which should've failed loudly with an exception, instead simply swallowed it and allowed the program to carry on in blissful ignorance of its impending doom.

The coding standard likely has a case for this: Always throw the exception, or always log the exception to a standard diagnostic stream or both, or .... but, again, without enforcement, it's just a paper document that nobody reads.

With Roslyn, you can build a diagnostic that catches this and even (when configured to do so) works with Visual Studio Team Foundation Server to prevent this code from ever being checked in until that empty catch block is fixed.

## Roslyn Diagnostics

As of this writing, project Roslyn is a preview release, installed as part of Visual Studio 2015 Preview. Once the Visual Studio 2015 Preview SDK and Roslyn SDK templates are installed, diagnostics can be written using the provided Extensibility template, Diagnostic with Code Fix (NuGet + VSIX). To start, as shown in **Figure 1**, select the diagnostic template and name the project EmptyCatchDiagnostic.

The second step is to write a Syntax Node Analyzer that walks the Abstract Syntax Tree (AST), looking for empty catch blocks. A tiny AST fragment is shown in **Figure 2**. The good news is the Roslyn compiler walks the AST for you. You need only provide code to analyze the nodes of interest. (For those familiar with classic "Gang-of-Four" design patterns, this is the Visitor pattern at work.) Your analyzer must inherit from the abstract base class DiagnosticAnalyzer and implement these two methods:

```
public abstract
  ImmutableArray<DiagnosticDescriptor> SupportedDiagnostics { get; }

public abstract void Initialize(AnalysisContext context);
```

The SupportedDiagnostics method is a simple one, returning a description of each analyzer you're offering up to Roslyn. The Initialize method is where you register your analyzer code with Roslyn. During initialization you provide Roslyn with two things: the kind of nodes in which you're interested; and the code to execute when one of these nodes is encountered during compilation. Because Visual Studio performs compilation in the background, these calls will occur while the user is editing, providing immediate feedback on possible errors.

> By creating an analyzer, you can ask Roslyn to enforce additional, higher-order kinds of "rules," helping to eliminate bugs without having to run additional tools.

Start by modifying the pre-generated template code into what you need for your empty catch diagnostic. This can be found in the source code file DiagnosticAnalyzer.cs within the EmptyCatchDiagnostic project (the solution will contain additional projects you can safely ignore). In the code that follows, what you see in **boldface** are the changes in relation to the pre-generated code. First, some strings describing our diagnostic:

```
internal const string Title = "Catch Block is Empty";
internal const string MessageFormat =
  "'{0}' is empty, app could be unknowingly missing exceptions";
internal const string Category = "Safety";
```

The generated SupportedDiagnostics method is correct; you only have to change the Initialize method to register your custom-written syntax analysis routine, AnalyzeSyntax:

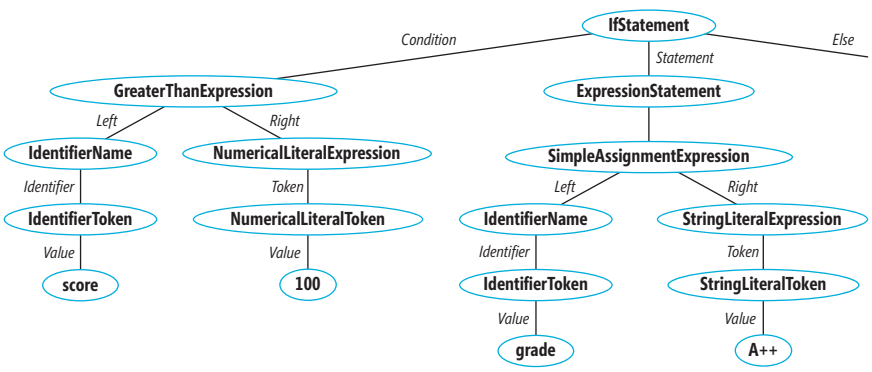



Figure 2 **Roslyn Abstract Syntax Tree for the Code Fragment: if (score > 100) grade = "A++";**

Figure 3 **Encountering a Catch Clause**

```
// Called when Roslyn encounters a catch clause.
private static void AnalyzeSyntax(SyntaxNodeAnalysisContext context)
{
  // Type cast to what we know.
  var catchBlock = context.Node as CatchClauseSyntax;

  // If catch is present we must have a block, so check if block empty?
  if (catchBlock?.Block.Statements.Count == 0)
  {
    // Block is empty, create and report diagnostic warning.
    var diagnostic = Diagnostic.Create(Rule,
      catchBlock.CatchKeyword.GetLocation(), "Catch block");
    context.ReportDiagnostic(diagnostic);
  }
}
```

```
public override void Initialize(AnalysisContext context)
{
  context.RegisterSyntaxNodeAction<SyntaxKind>(
    AnalyzeSyntax, SyntaxKind.CatchClause);
}
```

As part of the registration, note that you inform Roslyn you're only interested in *catch clauses* within the AST. This cuts down on the number of nodes fed to you and also helps keep the analyzer clean, simple and single-purposed.

During compilation, when a catch clause node is encountered in the AST, your analysis method AnalyzeSyntax is called. This is where you look at the number of statements in the catch block, and if that number is zero, you display a diagnostic warning because the block is empty. As shown in **Figure 3**, when your analyzer finds an empty catch block, you create a new diagnostic warning, position it at the location of the catch keyword and report it.

The third step is to build and run the diagnostic. What happens next is really interesting, and makes sense once you think about it. You just built a compiler-driven diagnostic—so how do you test it? By starting up Visual Studio, installing the diagnostic, opening a project with empty catch blocks and seeing what happens! This is depicted in **Figure 4**. The default project type is a VSIX installer, so when you "run" the project, Visual Studio starts up another instance of Visual Studio and runs the installer for it. Once that second instance is up, you can test it. Alas, automated testing of diagnostics is a bit beyond the scope of the project for now, but if the diagnostics are kept simple and single-focused, then it's not too hard to test manually.

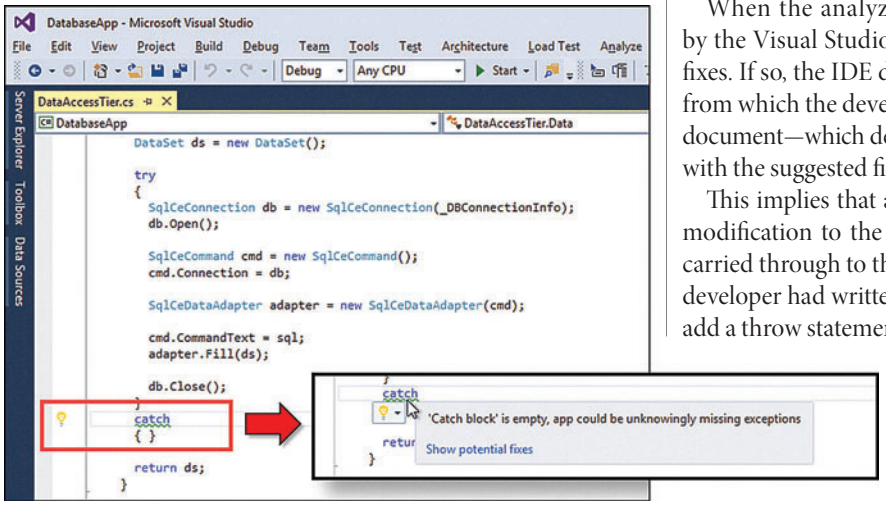



Figure 4 **Visual Studio Running an Empty Catch Block Diagnostic in Another Instance of Visual Studio**

## Don't Just Stand There, Fix It!

Unfortunately, a tool that points out an error that it could easily fix—but doesn't—is really just annoying. Sort of like that cousin of yours who watched you struggle to open the door for hours before deciding to mention that it's locked, then watched you struggle to find another way in for even longer before mentioning that he has the key.

Roslyn doesn't want to be that guy.

A code fix provides one or more suggestions to the developer—suggestions to hopefully fix the issue detected by the analyzer. In the case of an empty catch block, an easy code fix is to add a throw statement so that any exception caught is immediately rethrown. **Figure 5** illustrates how the code fix appears to the developer in Visual Studio, as a familiar tooltip.

> In the case of an empty catch block, an easy code fix is to add a throw statement so that any exception caught is immediately rethrown.

In this case focus your attention on the other pre-generated source file in the project, CodeFixProvider.cs. Your job is to inherit from the abstract base class CodeFixProvider and implement three methods. The key method is ComputeFixesAsync, which offers suggestions to the developer:

```
public sealed override async Task ComputeFixesAsync(CodeFixContext context)
```

When the analyzer reports an issue, this method is called by the Visual Studio IDE to see if there are any suggested code fixes. If so, the IDE displays a tooltip containing the suggestions, from which the developer may select. If one is selected, the given document—which denotes the AST for the source file—is updated with the suggested fix.

This implies that a code fix is nothing more than a suggested modification to the AST. By modifying the AST, the change is carried through to the remaining phases of the compiler, as if the developer had written that code. In this case, the suggestion is to add a throw statement. **Figure 6** is an abstract depiction of what's going on.

So your method builds a new subtree to replace the existing catch block subtree in the AST. You build this new subtree bottom up: a new throw statement, then a list to contain the statement, then a block to scope the list and, finally, a catch to anchor the block:

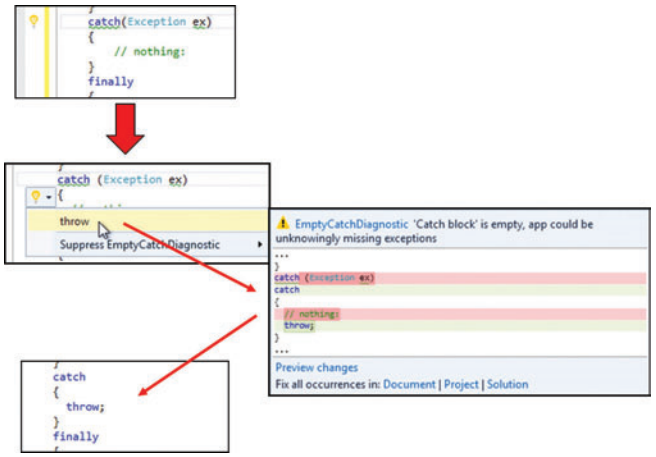



Figure 5 **A Code Fix Suggesting a Throw Within the Empty Catch Block**

```
public sealed override async Task ComputeFixesAsync(
  CodeFixContext context)
{
  // Create a new block with a list that contains a throw statement.
  var throwStmt = SyntaxFactory.ThrowStatement();
  var stmtList = new SyntaxList<StatementSyntax>().Add(throwStmt);
  var newBlock = SyntaxFactory.Block().WithStatements(stmtList);
  // Create a new, replacement catch block with our throw statement.
  var newCatchBlock = SyntaxFactory.CatchClause().WithBlock(newBlock).
    WithAdditionalAnnotations(
    Microsoft.CodeAnalysis.Formatting.Formatter.Annotation);
```

The next step is to grab the root of the AST for this source file, find the catch block identified by the analyzer and build a new AST. For example, newRoot denotes a newly rooted AST for this source file:

```
  var root = await context.Document.GetSyntaxRootAsync(
    context.CancellationToken).ConfigureAwait(false);

  var diagnostic = context.Diagnostics.First();
  var diagnosticSpan = diagnostic.Location.SourceSpan;

  var token = root.FindToken(diagnosticSpan.Start); // This is catch keyword.
  var catchBlock = token.Parent as CatchClauseSyntax; // This is catch block.

  var newRoot = root.ReplaceNode(catchBlock, newCatchBlock); // Create new AST.
```

The last step is to register a code action that will invoke your fix and update the AST:

```
  var codeAction =
    CodeAction.Create("throw", context.Document.WithSyntaxRoot(newRoot));
  context.RegisterFix(codeAction, diagnostic);
}
```

For a variety of good reasons, most data structures in Roslyn are immutable, including the AST. This is a particularly good choice here, because you don't want to update the AST unless the developer actually selects the code fix. Because the existing AST is immutable, the method returns a new AST, which is substituted for the current AST by the IDE if the code fix is selected.

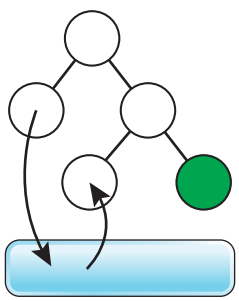


Figure 6 **Updating the Abstract Syntax Tree**

You might be concerned that immutability comes at the high cost of memory consumption. If the AST is immutable, does that imply a complete copy is needed every time a change is made? Fortunately, only the differences are stored in the AST (on the grounds that it's easier to store the deltas than to deal with the concurrency and consistency issues that making the AST entirely mutable would create) to minimize the amount of copying that occurs to ensure immutability.

## Breaking New Ground

Roslyn breaks some new ground by opening up the compiler (and the IDE, as well!) this way. For years, C# has touted itself as a "strongly typed" language, suggesting that up-front compilation helps reduce errors. In fact, C# even took a few steps to try to avoid common mistakes from other languages (such as treating integer comparisons as Boolean values, leading to the infamous "if (x = 0)" bug that often hurt C++ developers). But compilers have always had to be extremely selective about what rules they could or would apply, because those decisions were industry-wide, and different organizations often had different opinions on what was "too strict" or "too loose." Now, with Microsoft opening up the compiler's innards to developers, you can begin to enforce "house rules" on code, without having to become compiler experts on your own.

With Microsoft opening up the compiler's innards to developers, you can begin to enforce "house rules" on code, without having to become compiler experts on your own.

Check out the Roslyn project page at roslyn.codeplex.com for details on how to get started with Roslyn. If you want to dive more deeply into parsing and lexing, numerous books are available, including the venerable "Dragon Book," officially published as "Compilers: Principles, Techniques & Tools" (Addison Wesley, 2006) by Aho, Lam, Sethi and Ullman. For those interested in a more .NET-centric approach, consider "Compiling for the .NET Common Language Runtime (CLR)" (Prentice Hall, 2001) by John Gough, or Ronald Mak's "Writing Compilers and Interpeters: A Software Engineering Approach" (Wiley, 2009).

Happy coding! ■

**Ted Neward** *is the CTO at iTrellis, a consulting services company. He has written more than 100 articles and authored 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 or ted@itrellis.com if you're interested.*

**Joe Hummel, Ph.D.** *is a research associate professor at the University of Illinois, Chicago, a content creator for Pluralsight, a Visual C++ MVP, and a private consultant. He earned a Ph.D. at UC Irvine in the field of high-performance computing and is interested in all things parallel. He resides in the Chicago area, and when he isn't sailing can be reached at joe@joehummel.net.*

**Thanks** to the following Microsoft technical expert for reviewing this article: *Kevin Pilch-Bisson*

# NAVIGATE THE .NET HIGHWAY

## TRACKS INCLUDE:

- Visual Studio/.NET
- JavaScript/HTML5 Client
- ASP.NET
- Mobile Client
- Windows Client
- Database and Analytics
- Cloud Computing
- SharePoint/Office


## REGISTER NOW AND SAVE $300!

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

Use promo code VSLFEB2

**vslive.com/sf**

## Join us on the Ultimate Code Trip in 2015!

| LAS VEGAS Code Trip | AUSTIN Code Trip | SAN FRANCISCO Code Trip | REDMOND Code Trip | NEW YORK Code Trip | ORLANDO Code Trip |
|---|---|---|---|---|---|
| March 16-20 | June 1-4 | June 15-18 | August 10-14 | Sept. 28- Oct. 1 | Nov. 16-20 |

**CONNECT WITH VISUAL STUDIO LIVE!**

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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



RACHEL APPEL

# Implement Search in Windows Store and Windows Phone Store Apps

Search is an integral part of our digital life. Millions of people use sites like Bing and Google to find information every day. Millions more search through the Web, apps and proprietary data stores alike. Because search is arguably the most frequently used digital feature, the smart move is to help users more easily search with your apps.

## How Windows Facilitates Search

Due to the physical form factor differences of PCs, tablets and laptops, the Windows OSes provide search UIs that behave accordingly. For example, Windows Phone 8.1 uses Cortana to search (read more about Cortana at bit.ly/1nFGMxG). Apps can perform a custom search from within a Windows Phone Store app with or without Cortana. Windows Phone-based devices have a special hardware button for search, located on the bottom right of the phone itself, next to the Windows (center) and Back (left) buttons.

You can do a number of things to invoke search with Windows 8. The easiest way is to just start typing while you're on the Windows Start screen. That kicks off the global search, across the device and the Internet. Using a swipe touch gesture is another way to invoke search for touch-enabled devices. You could also jam the mouse into the top-right or bottom-right corners of the screen to initiate search.

> When a user sees a magnifying glass, he knows it always means search.

Finally, if you live and die by shortcut keys, you can also use Windows+S to display the Windows Search charm. Iconography doesn't need to change due to form factors. When a user sees a magnifying glass, he knows it always means search.

Microsoft recommends using the SearchBox control, although it's fine to use the Search charm (also called a Search Contract) for backward compatibility. You can read more about that at bit.ly/1xkqwXN. When using the SearchBox control, you can add it to your app's canvas or show it in the app bar. If search is one of the primary methods of interacting with your app, it's best to present it consistently throughout the app UI.

A good location is anywhere easily spotted on the app's canvas, usually the top-right corner. Apps that deal with news, movies, sports, academic papers and financial reports are great examples of apps that need a prominent search box. Sometimes users like to browse lazily, but more often they know what they want and prefer to navigate directly to it via search.

You might be concerned with a search box taking up too much screen real estate. That's a valid concern. Using a revealing search

Figure 1 **The SearchBox with Suggestions Using XAML and C#**

```
<SearchBox x:Name="SearchBox" Height="35"
  HorizontalAlignment="Stretch" VerticalAlignment="Bottom"
  SuggestionsRequested="SearchBoxEventsSuggestionsRequested"
  QuerySubmitted="SearchBoxEventsQuerySubmitted"
  FocusOnKeyboardInput="True"/>
public sealed partial class SearchBoxWithSuggestions :
  SDKTemplate.Common.LayoutAwarePage
{
  public SearchBoxWithSuggestions()
  {
    this.InitializeComponent();
  }
  private static readonly string[] suggestionList =
  {
    "ActionScript", "Ada", "Basic", "C", "C#", "C++", "Clipper",
      "Clojure", "COBOL", "ColdFusion", "Dart", "Delphi", "Erlang",
      "F#", "Haskell", "HTML", "Java", "JavaScript", "LISP",
      "Objective-C", "Pascal", "Perl", "PowerShell", "R", "Ruby",
      "Rust", "RPG", "SQL", "SmallTalk", "Small Basic", "Swift",
      "TypeScript", "Turbo C", "Visual Basic"
  };

  private void SearchBoxEventsSuggestionsRequested(
    object sender, SearchBoxSuggestionsRequestedEventArgs e)
  {
    string queryText = e.QueryText;
    if (!string.IsNullOrEmpty(queryText))
    {
      Windows.ApplicationModel.Search.
        SearchSuggestionCollection suggestionCollection =
        e.Request.SearchSuggestionCollection;
      int n = 0;
      foreach (string suggestion in suggestionList)
      {
        if (suggestion.StartsWith(queryText,
          StringComparison.CurrentCultureIgnoreCase))
        {
          Uri uri = new Uri("ms-appx:///Assets/laptop1.png");
          RandomAccessStreamReference imageSource =
            RandomAccessStreamReference.CreateFromUri(uri);
          suggestionCollection.AppendResultSuggestion(
            suggestion, "", n.ToString(), imageSource, suggestion);
        }
      }
    }
  }
  private void SearchBoxEventsQuerySubmitted(
    object sender, SearchBoxQuerySubmittedEventArgs e)
  {
    SearchListView.Items.Add(e.QueryText);
  }
}
```

Figure 2 **The SearchBox with Suggestions Using HTML and JavaScript**

```
<div id="searchBox" data-win-control="WinJS.UI.SearchBox"></div>
(function () {
"use strict";
var suggestionList = ["ActionScript", "Ada", "Basic", "C", "C#", "C++", "Clipper",
  "Clojure", "COBOL", "ColdFusion", "Dart", "Delphi", "Erlang", "F#", "Haskell",
  "HTML", "Java", "JavaScript", "LISP", "Objective-C", "Pascal", "Perl",
  "PowerShell", "R", "Ruby", "Rust", "RPG", "SQL", "SmallTalk",
  "Small Basic", "Swift", "TypeScript", "Turbo C", "Visual Basic"];
var page = WinJS.UI.Pages.define("/html/S1-SearchBoxWithSuggestions.html", {
    ready: function (element, options) {
      var searchBox = document.getElementById("searchBox");
      searchBox.addEventListener("suggestionsrequested",
        suggestionsRequestedHandler);
      searchBox.addEventListener("querysubmitted", querySubmittedHandler);
      searchBox.winControl.focusOnKeyboardInput = true;
    }
  });
function suggestionsRequestedHandler(eventObject) {
    var queryText = eventObject.detail.queryText,
      query = queryText.toLowerCase(),
      suggestionCollection = eventObject.detail.searchSuggestionCollection;
    if (queryText.length > 0) {
      for (var i = 0, len = suggestionList.length; i < len; i++) {
        if (suggestionList[i].substr(0, query.length).toLowerCase() == query) {
          suggestionCollection.appendQuerySuggestion(suggestionList[i]);
        }
      }
    }
  }
function querySubmittedHandler(eventObject) {
    var queryText = eventObject.detail.queryText;
    WinJS.log && WinJS.log(queryText, "sample", "status");
  }
})();
```

icon is one way to show a small but noticeable visual search indicator. After a user clicks or taps on the revealing search glyph, the SearchBox reveals itself so they can enter a search string and view results.

You can use the SearchBox to find data locally or globally. As you might expect, global searches are when your app is accessing data outside the app itself. This can be on a removable device, network or the Internet. If you do search through files such as music or pictures on the device, don't forget to set the capabilities in the program's manifest.

## Implement Search with the SearchBox Control

You can incorporate search boxes into your apps using either XAML or HTML for Windows Store and Windows Phone Store apps. The controls and API calls conceptually work the same across languages, but naturally with different syntax. Coding search for the search charm is also roughly the same as far as complexity is concerned. The code must perform the same basic steps, and the same UX guidelines apply, regardless of language.

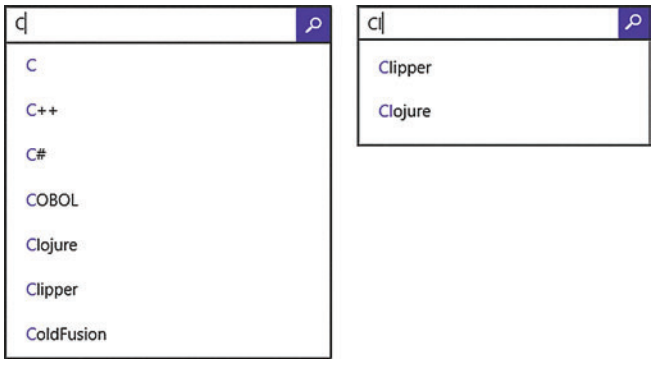



Figure 3 **Three SearchBoxes with Query and Result Suggestions**

**Figure 1** demonstrates a XAML SearchBox. You can put this control within any container control such as a StackPanel. As with other controls, you must wire up events that fire in response to the user invoking search. When you use the SearchBox control, there's no need to set the search icon. You'll likely want to set the FocusOnKeyboardInput to True. That lets users simply start typing to give focus control to the SearchBox, which makes for an easier search experience.

In XAML, the SearchBoxEventsSuggestionsRequested and SearchBoxEventsQuerySubmitted events are the two primary event wire-ups you'll need. SearchBoxEventsSuggestionsRequested fires once for each keystroke entered to capture the keystrokes in real time. As its name suggests, SearchBoxEventsQuerySubmitted happens when the user presses Enter, clicks, taps on the search icon or otherwise triggers a search. The SearchBoxEventsQuerySubmitted event is where you add code to perform the actual search. **Figure 1** shows both events in action.

In C#, the first thing you need to do is supply a list of search strings to use as suggestions. **Figure 1** shows an array of strings named suggestionList. The list contains the names of several programming languages. The code in **Figure 1** demonstrates a search implementation with the SearchSuggestionCollection. When the SearchBoxEventsSuggestionsRequested event fires, its argument named "e" contains the SearchSuggestionCollection to which you can append queries. That's reflected in **Figure 1** when the suggestionCollection variable is declared and set.

You can append queries to the SearchSuggestionCollection through the AppendQuerySuggestion, AppendQuerySuggestions, AppendSearchSuggestion or AppendResultSuggestion methods. Result suggestions appear in the same list as query suggestions, but they let you pass in extra data such as an image into the SearchBox.

While XAML has the notion of a Resource to set styles and aesthetic features, HTML uses CSS to perform these tasks. As an example, because the  element in **Figure 2** contains no reference to any styles, it will use the default Windows Library for JavaScript (WinJS) .win-searchbox class that's part of the WinJS base CSS.

**Figure 3** shows the runtime results of either **Figure 1** or **Figure 2**.

Notice the SearchBoxes in **Figure 3** show filtered suggestions based on what the user enters. This is a great feature. Fortunately, it's something the SearchBox control does for you automatically in XAML or HTML. However, you must perform the actual search yourself in the query submission event. That means it's up to you to read the files, access the databases and Web services, or search the Web.

When you append items to the SuggestionCollection, the AppendResultSuggestion method lets you supply more information than the AppendQuerySuggestion method. Pass in the text, description, image and alternate text to apply items in the list, as the code in **Figure 4** reveals. **Figure 5** illustrates the runtime screenshot the code in **Figure 4** will create.

Figure 4 **Code to Add Images to Suggestions**

```
private void SearchBoxEventsSuggestionsRequested(object sender,
  SearchBoxSuggestionsRequestedEventArgs e)
{
  string queryText = e.QueryText;
  if (!string.IsNullOrEmpty(queryText))
  {
    Windows.ApplicationModel.
      Search.SearchSuggestionCollection suggestionCollection =
      e.Request.SearchSuggestionCollection;
    int n = 0;
    foreach (string suggestion in suggestionList)
    {
      if (suggestion.StartsWith(queryText,
        StringComparison.CurrentCultureIgnoreCase))
      {
        Uri uri = new Uri("ms-appx:///Assets/laptop.png");
        RandomAccessStreamReference imageSource =
          RandomAccessStreamReference.CreateFromUri(uri);
        suggestionCollection.AppendResultSuggestion(
          suggestion, "", n.ToString(), imageSource, suggestion);
      }
    }
  }
```

The image argument passed to the AppendResultSuggestions class is an IRandomAccessStreamReference type from the Windows.Storage.Streams namespace. If you're using JavaScript, you'll have to create a Uri using the same CreateFromUri method. This is in contrast to the usual way of setting an image in HTML with a src attribute.

At this point, you have a working SearchBox and suggestions with text and images. The next step is to display the search results. In both XAML and HTML, you can add a pre-defined search results page for displaying and interacting with the results. Visual Studio provides page templates with code that displays search results with a filtered list of data that you provide. Because these pages are customizable, you can show the results exactly how you want.

When you add a SearchResultsPage from the New File dialog in Visual Studio, it creates a page with a ListView for displaying the search results. For more information on using the ListView, see my December 2013 column, "Everything You Need to Know About the ListView Control" (msdn.microsoft.com/magazine/dn532209). Of course, it's not mandatory you use the search results page template—you can incorporate search results anywhere into the UI that makes sense and is easiest for the user. When you do, make sure you check out the UX guidelines first.

## Search UX Guidelines

If the user can never find anything, or has difficulty searching, he'll be more apt to rate your app poorly in the store. Even worse is when a user won't buy your app because search in the trial version doesn't work.

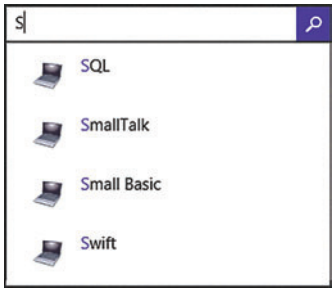



Figure 5 **SearchBox with Result Suggestions and Images**

Because search is such a frequently used feature, take the time to get it right in both paid and trial apps. Here are some suggestions about how to implement search in your apps and raise those ratings:

- If search is a frequently used feature of your app, place it where users can immediately find and use it.
- Help users with query and result suggestions. Users rely on suggestions to quickly navigate throughout the app and perform actions.
- Display results so they're easy to skim. Aggregate information is the friend of both you and the user. The point of search is to present information to users so they can make a choice as to what details to pursue.
- Ensure the search box works with a touch keyboard, as well as physical keyboards.
- Support Ctrl+F as the keyboard shortcut for finding text in your app (Windows only).

Many of these points are aesthetic in nature. If anything, Windows UX guidelines err on the side of presenting fewer but more important pieces of information. A user should be able to navigate back to his previous location from the search results through a back button. To catch up or refresh your knowledge on navigation in Windows Store apps, read my August 2013 column, "Navigation Essentials in Windows Store Apps" (msdn.microsoft.com/magazine/dn342878).

You should always provide search suggestions, especially on the phone. No user wants to enter a search query and receive no further help. It's also much harder to type quickly and correctly on small devices. Phone users have fewer or restricted methods of input. Having to tap out entire words just makes your app harder to use and frustrates the user.

> If the user can never find anything, or has difficultly searching, he'll be more apt to rate your app poorly in the store.

## Searching for the Conclusion

As you can see, implementing a pleasant search experience is easy to do for Windows Store and Windows Phone Store apps. Adding search capabilities not only makes your app more robust, but it offers users easy access to an important and frequently used feature. Keep in mind you can search both global and local data.

Don't forget to review and follow the guidelines outlined in this article and the ones Microsoft describes at bit.ly/1BQ5fGZ when implementing search in your app. ■

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

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



# 5 Years Down the Road

How the hell did five years go by? I swear I just submitted the first installment of this column, entitled "The Human Touch." In it I argued that geeks fundamentally misunderstand their users, as evidenced by their terrible designs. I chided them: "Humans are not going to stop being human any time soon, no matter how much you might wish they would evolve into something more logical. Good applications recognize this and adjust to their human users, instead of hoping, futilely, for the opposite."

That principle certainly applies today. But my lying calendar insists that I did it in February 2010.

It was a different world then. Windows 7 still had four months before RTM. The now-ubiquitous mobile sector didn't exist. The iPhone was only a year old, and the most popular app was the picture of a full beer stein, in which the level went down when pathetic geeks tilted their phones to their lips. The cloud meant Hotmail. My daughters still believed in Santa Claus.

It's been quite a journey from then to now. I salute Keith Ward, the editor who recruited me for this gig before wisely fleeing to VisualStudioMagazine.com. He inflicted me on his successor, Michael Desmond, whose life I make quite interesting. And I salute Microsoft, for taking (almost) all the lumps I've dished out to them without squawking (much), considering it owns this magazine.

Above all, I've loved meeting you, my readers—through e-mails, article comments and in person at conferences. You are the reason I write, to celebrate the good and castigate the bad, and to let in the sunshine. Gadfly, jester, cynic, curmudgeon, loudmouth, conscience, jerk; call me what you will. I'm glad you're along for the ride, because it sure would suck doing this all by myself.

"Your column is always the first page I turn to," say some readers. "I use it to line my parakeet's cage, with your picture facing up," say others. I feel especially honored by the reader who said, "I can give this column to my parents, so they can understand something about what I do."

Sometimes these columns have been painful to write, like the one about Lloyd's daughter (aka.ms/k3so4r). And I feel like I spent much of 2011 eulogizing late industry titans. Ken Olsen (aka.ms/czi7w5). Dennis Ritchie and Steve Jobs (aka.ms/xzape5). Even Simba (aka.ms/ar9yxu).

## Good Advice and Bad Intentions

This much I'll say: Microsoft has done well by following my advice. It's doing a decent job of shedding its PC blindness, as I urged back in my June 2011 column (aka.ms/d417jm). More recently, my November 2013 column, "Advice to the New CEO," called for Redmond to start becoming a cloud and services company (aka.ms/pjuinu). Microsoft is moving well in that direction today, faster than I would have thought possible, though it's still finding mobile devices to be tricky.

Not all my insights have been on target. I wonder what I'll be writing for my next "Biggest Misteaks" column (aka.ms/ktri5g). Last month, I described my 2007 prediction that the "iPhone would crash in flames." That prognostication will be hard to top, but you can count on me to keep trying. Right now, I'd be lying if I said I saw the point of wearables, like Google Glass and the smart watches now coming on the market.

Gadfly, jester, cynic, curmudgeon, loudmouth, conscience, jerk; call me what you will. I'm glad you're along for the ride, because it sure would suck doing this all by myself.

The technology changes quickly, but our need to understand its impact on our users and our world doesn't. Nor does our need to step back and have a good look at ourselves, with laughter and tears, as needed. I promise you, I won't change that. I promise you, I'll keep calling 'em as I see 'em, pouring oil on troubled fires, for your instruction, amusement and thought stimulation. I promise, I'll keep doing as Robert Heinlein's character Lazarus Long said: "Here's one monkey that's going to keep on climbing, and looking around him to see what he can see, as long as the tree holds out." ■

**DAVID S. PLATT** *teaches programming .NET at Harvard University Extension School and at companies all over the world. He's the author of 11 programming books, including "Why Software Sucks" (Addison-Wesley Professional, 2006) and "Introducing Microsoft .NET" (Microsoft Press, 2002). Microsoft named him a Software Legend in 2002. He wonders whether he should tape down two of his daughter's fingers so she learns how to count in octal. You can contact him at rollthunder.com.*

**All New Releases!**


# .NET Controls for the Professional Developer

ComponentOne delivers Data Visualization, UI, Spreadsheet and Reporting Controls for Microsoft Visual Studio Development

| Reporting | Data Visualization | Spreadsheet |
|---|---|---|

### ActiveReports

Easily create a variety of form-based, analytical and transaction reports. Visualize data with graphical components, design reports in the IDE & extend the reach of your reports with versatile viewers.

### Studio Enterprise

Hundreds of data and UI controls for all .NET platforms including grids, charts, reports and schedulers. Make building next-gen UIs in today's apps easier with an array of samples with source code.

### Spread Studio

A Microsoft Excel-like spreadsheet control with the functionally of an advanced data grid, for adding power to any .NET application. Includes a formula engine and flexible tabular layout.

Download your free trials at
**ComponentOne.com**

Visual Studio Partner

ComponentOne a division of GrapeCity

18TH SUCCESSFUL YEAR!

FROM THE ORGANISERS OF **SOFTWARE ARCHITECT**

# DEVWEEK2015

**Monday 23 – Friday 27 March | Central Hall, Westminster, London**

## THE DEFINITIVE DEVELOPER CONFERENCE

**World-renowned speakers, including**

ALLEN HOLUB

KEVLIN HENNEY

DINO ESPOSITO

NEAL FORD

**and many more**

**Keep your whole development team's skills up to date with the latest technologies, best practices and frameworks**

- **23 FULL-DAY WORKSHOPS**
- **105 90-MINUTE BREAKOUT SESSIONS**

BOOK BY FRIDAY 30 JANUARY 2015
**SAVE UP TO £200**

**GOLD SPONSORS:**

amazon web services | intel Software

**SPONSORS:**

Alachisoft | ASPOSE Your File Format APIs | IncrediBuild BEYOND ACCELERATION | K2

**MEDIA PARTNERS:**

BIG DATA TALENT | BIG DATA VENDORS | compare the cloud.net | CloudCow.com | datanami | ENTERPRISE MANAGEMENT 360° | tuts+

Agile | Architecture | BI | Big Data | Cloud | Database | DevOps | IoT | Leadership | Mobile | MQ | MS Tech
Patterns | Programming Languages / Techniques | Security | Software Design | Testing | UI/UX | Web

www.devweek.com

# DON'T MISS THE UK'S LEADING DEVELOPER CONFERENCE

**DevWeek 2015 is the UK's number one destination for software developers, architects and analysts. With insights on the latest technologies, best practice and frameworks from industry-leading experts, plus hands-on workshop sessions, DevWeek is your chance to sharpen your skills – and ensure every member of your team is up to date.**

**23 full-day workshops**
**105 90-minute breakout sessions**
**49 expert speakers**
**Plus keynote presentations from Allen Holub and Kevlin Henney**

## VENUE


### IN THE HEART OF LONDON

**For five full days, from Monday 23 March to Friday 27 March, DevWeek takes over Westminster's iconic Central Hall.**

**Built in 1912, Central Hall is one of the UK's oldest purpose-built conference centres. Offering easy access to public transport and the restaurants and theatres of the West End, the venue fuses architectural grandeur with state-of-the-art facilities – making it the ideal setting for the UK's premier developer conference.**

**For more information about Central Hall, visit www.c-h-w.com**

**NEAREST TUBE STATIONS:**

**Westminster**
(District, Circle & Jubilee)
**St James's Park**
(District & Circle)

# ENJOY YOUR CONFERENCE, YOUR WAY

With shareable tickets and online catch-up, DevWeek gives you all the information you need, when you want it – ensuring you and your team make the most of every session.

**Share your ticket**

Only have time to attend one day? Get great value out of DevWeek by sharing your ticket with others in your team. That way, you can make the most of five packed days of sessions and workshops. Our online registration page lets you add colleagues' details so we know who will be joining us for each day of DevWeek.

**Never miss a session**

So much content is packed into DevWeek's five days, there are bound to be times when you wish you could be in two places at once. But you needn't miss out: all our sessions are filmed (subject to speaker approval) – and as a registered delegate, you'll have exclusive access to the whole event online to watch when you want.

**Topics**

- Agile
- Architecture
- BI
- Big Data
- Cloud
- Database
- DevOps
- IoT
- Leadership
- Mobile
- MQ
- MS Tech
- Patterns
- Programming Languages / Techniques
- Security
- Software Design
- Testing
- UI/UX
- Web

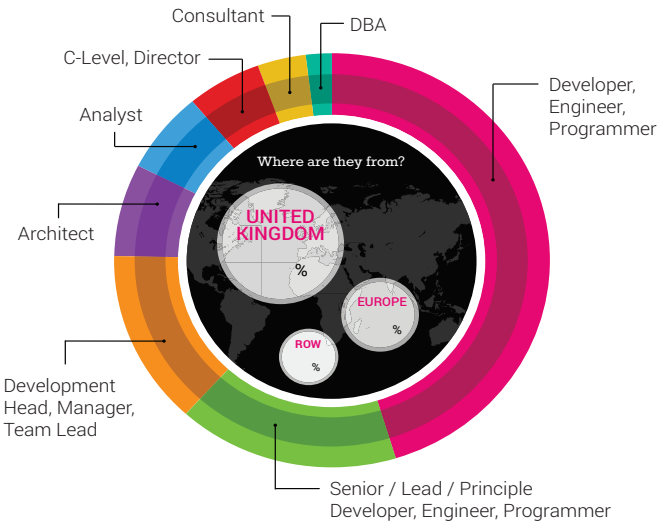
## SPONSORSHIP & EXHIBITION

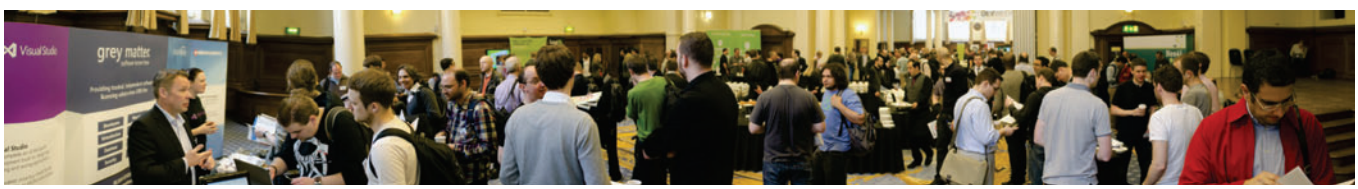
### Do you provide services or technologies to the developer community? Let DevWeek work for you

DevWeek is the UK's leading event for software developers, DBAs and IT architects. As an exhibitor or sponsor, we can can help you reach an engaged audience of professionals with a package that's custom-designed to suit your needs.

To discuss the wide range of sponsorship and exhibition opportunities available, contact **Chris Handsley**
**+44 (0)207 830 3634**
**chris.handsley@publicis-blueprint.co.uk**

### WHO WILL YOU MEET AT DEVWEEK?



**GOLD SPONSORS:**

amazon web services
intel Software

**SPONSORS:**

Alachisoft
ASPOSE Your File Format APIs
IncrediBuild BEYOND ACCELERATION
K2

**MEDIA PARTNERS:**

BIG DATA TALENT
BIG DATA VENDORS
compare the cloud.net
CloudCow.com
datanami
ENTERPRISE MANAGEMENT 360°
tuts+

## PRE-CONFERENCE WORKSHOPS

**DevWeek's programme of pre-conference workshops covers a wide range of subjects, from Agile systems to app development.**

**All workshops run for a full day, from 09.30 to 17.30, with short breaks in the morning and afternoon, and a lunch break at 13.00.**

**09.30 - 17.30**

**ALLEN HOLUB**

### AGILE/DESIGN FROM START TO FINISH

Agile systems are, by necessity, tightly coupled to the user's notion of what the system is doing. Without that connection, the software can't stand up to the stress of constant change that all Agile processes mandate. Moreover, the process you use influences the architecture of your system. That's why hybrid processes that mix Agile and traditional practices often fail. The hybrid architectures that come out of those processes are unworkable.

In this workshop, Allen will talk about both process and architecture. We'll look at how Agile processes work, and see how both architecture and low-level design naturally fall out of those processes. Specifically, we'll examine the role of stories: how they're created and developed, and how they flow through the process, with a focus on developing an optimal architecture that closely mirrors both the stories themselves and the assumptions that underlie the stories.

We'll also work through a real-world example of the process from requirements gathering and problem-statement definition, to story development (use-case analysis) and the simultaneous construction of lightweight dynamic and static models that underlie the code.

**WORKSHOP REF: DW01**

**PEARL CHEN**

### INTERNETTING YOUR THINGS

From big consumer success stories, such as the Nest Thermostat, to more offbeat monitoring systems, such as Botanicalls (which lets your plant call you when it is thirsty), your *things* are finding their own voice through small but powerful embedded microcontrollers. The market for the Internet of Things (IoT) and wearables is exploding. But what are your options for getting started with making physical things when you're more used to writing software? In this workshop, Pearl explores a few hardware options that are great for hobbyist and rapid prototyping.

You'll get hands-on time to choose from some of the available hardware platforms, from a standard Arduino to an Intel Edison, MaKey MaKey, or LightBlue Bean. You'll be given a self-paced guide on how to set up the development environment on your computer and get a "Hello World" program running (typically an onboard blinking light). Once you have a feel for the technology, we'll go through a project brainstorming exercise. Finally, you'll be let loose to build your own hardware-based prototype in groups of 2-4. The workshop will be capped at 30 participants so expect a very team-oriented day.

**WORKSHOP REF: DW02**

**SAHIL MALIK**

### JAVASCRIPT: BEYOND THE BASICS

In this workshop, Sahil will teach you advanced concepts in JavaScript, allowing you to structure your code well and learn JavaScript beyond the basics we have been tinkering with.

This workshop teaches you not to just *know* JavaScript, but to be *good at* JavaScript. There are plenty of hands-on labs, which will walk you through objects, prototypes, scopes, this variable, debugging, performance and best practices.

We will begin with the very basics of JavaScript: variables, language syntax, referencing files, loops, conditions, built-in functions and custom functions, arrays and so on. We'll move on to talk about functions as expressions, closures to structure your code, the concept of nested scopes, and the confusion around "this". We'll learn all about objects and prototypes, and how you can mimic things in JavaScript that are usually reserved for higher-level languages.

We'll solidify that knowledge with some best practices and debugging tricks. We'll also explain how to avoid common pitfalls, and how to organise your code in modules to keep it maintainable and understandable. Lastly, there's performance – let's not forget performance!

**WORKSHOP REF: DW03**

**JAMES MONTEMAGNO**

### REUSING YOUR .NET AND C# SKILLS TO DEVELOP NATIVE APPS FOR WINDOWS, IOS AND ANDROID

With .NET, Xamarin and portable class libraries, you can now create native apps that target iOS, Android and Windows without compromising on performance, user experience or developer productivity. And you can do it all within Visual Studio IDE or Xamarin Studio IDE for Mac or PC.

James will introduce the Xamarin platform, including building native iOS and Android apps with C#. We'll look at the fundamentals of how each platform works, and deep-dive into platform-specific functionality. Designing iOS and Android apps could not be easier with Xamarin's integrated iOS and Android designers for both Visual Studio and Xamarin Studio. You'll find out how to use both of these designers to craft unique user interfaces for each platform. We'll also take a look at sharing code with Universal Windows apps, enabling you reach all platforms from a single shared C# code base.

You'll gain a full understanding of the Xamarin platform and how to build iOS and Android apps using C#, as well as a solid introduction to code reuse techniques, plus lots of sample code to take home!

**WORKSHOP REF: DW04**

> "Excellent mix of subjects– would recommend unreservedly"
>
> SOFTWARE DEVELOPER 2014 DELEGATE

ROBERT SMALLSHIRE & AUSTIN BINGHAM

**THE POWER OF REVIEW**

Review by peers, colleagues, experts and stakeholders is perhaps the most effective tool we have for improving the quality of software. But if review is so wonderful, why is it used so infrequently?

In this workshop, Robert and Austin will show you how to conduct effective code, design and requirements reviews using a variety of techniques from the relatively informal sort of reviews you're perhaps doing already, through to the most formal inspections. We'll work together to understand what makes a good review, and help you to identify behaviours that lead to poor outcomes, in the form of either defective software or unhappy colleagues.

Throughout this workshop, you'll receive plenty of advice on how you can introduce effective technical reviews into your engineering culture.

**WORKSHOP**
**REF: DW05**

JULES MAY

**F#: THE LITTLE LANGUAGE WITH A LOT OF BITE**

Since it was released in 2008, F# has inspired more and more excitement. It's a functional language, like Haskell or ML, and yet it's built on top of .NET, so it leverages all the mature object-oriented power that the platform provides. The result is neither purely functional, nor obviously object-oriented, but is something entirely new: a unique and powerful mixture of the two.

Those programmers who have already discovered F# find they can write code that is shorter, faster, vastly more reliable, and delightfully reusable. This is no research language – it's being used on real, large-scale projects, and it has the backing of the F# Software Foundation and Microsoft.

If you want to understand what the fuss is about, this is the day for you. In this workshop, Jules will touch on what makes F# programming so special, and show where it gets its power from. We'll write some functional code, we'll write some of F#'s unique take on object-oriented code, and we'll write some code that only F# can do.

**WORKSHOP**
**REF: DW06**

IDO FLATOW

**SECURING ASP.NET APPLICATIONS AND SERVICES: FROM A-Z**

When you think of ASP.NET security, the first things that come to mind are Windows authentication and forms authentication using ASP.NET Membership. For years, those were the common authentication techniques for ASP.NET applications and services. But with the new releases to the ASP.NET Identity system, those days are long gone.

For the enterprise, ASP.NET broadened its support from the on-premises Active Directory to include Microsoft Azure Active Directory. By supporting external identity providers, such as Facebook, Microsoft Account and Twitter, the new ASP.NET Identity system makes the process of securing an application less scary than ever.

In this workshop, Ido will start from the basics of getting to know concepts such as SSL, OAuth, OpenID and claim-based authorisation. From there we will continue to explore the various scenarios of using self-managed identities, Active Directory and ADFS, external identity providers (Facebook, Google, Microsoft) and Microsoft Azure Active Directory.

**WORKSHOP**
**REF: DW07**

KLAUS ASCHENBRENNER

**SQL SERVER QUERY TUNING FOR DEVELOPERS**

Are you a developer writing T-SQL queries for SQL Server databases? Maybe you're already mastered the basics of T-SQL, but want to reach a higher level in T-SQL to write better performing queries? In this workshop, Klaus will take a full day to talk about how to improve your T-SQL skills to solve complex problems, and how to further speed up your queries by applying a good indexing strategy to your T-SQL queries.

In the workshop, we'll cover four modules: query processing basics (set theory, predicate logic, relational models and logical query processing), physical query processing (execution plans, data access paths, physical join operators, aggregation operators and spool operators), temporary data and aggregations (temp tables, table variables, common table expressions, and aggregations and pivoting), and working with windowing functions (window aggregate functions, ranking functions, distribution functions, offset functions, query tuning guidelines and parallelism optimisations).

**WORKSHOP**
**REF: DW08**

DINO ESPOSITO

**ONE DAY OF TWITTER BOOTSTRAP**

As a responsive web framework, Twitter Bootstrap is leading the world of web development today, setting new standards and capturing followers. In this workshop, Dino will provide a day-long tour of the library and delve deep into its HTML chunks, CSS styles and JavaScript components.

We'll focus on facilities available for building static and responsive layouts, rich input forms and advanced features such as auto-completion, modals, tabs, carousels and more. We'll also consider the downsides of the framework, missing pieces (ie. image handling) and its overall role in the broader context of responsive and device-friendly sites. This workshop is ideal for clearing up a few things you may already have heard about Bootstrap, or just for gaining an additional perspective about it. In any case, after this workshop you should be ready to get into it at any level of further complexity.

**WORKSHOP**
**REF: DW09**

SEB ROSE

**BDD BY EXAMPLE**

In this workshop, Seb will provide a practical introduction to using examples to specify software. You'll learn to break down complex business requirements with your stakeholders, using examples in their own language, giving you the tools you need to explore their ideas before you even write any software.

This workshop is for everybody involved in the process of developing software, so please bring product owners, testers and architects along. As well as describing what BDD is (and isn't), we'll spend a lot of time practising collaborative analysis to make sure that our stories are appropriately sized, easy to read and unambiguous. We'll develop a "ubiquitous language", explore the workings of the Three-Amigos meeting, and really get to grips with the slippery interaction between features, stories, acceptance criteria and examples.

We'll use pens, cards and other bits of paper, so you won't need to know any tools in advance, or even remember your laptop!

To build further on these ideas, don't miss Seb's post-conference workshop: *"Applied BDD with Cucumber, Cucumber-JVM and SpecFlow"*.

**WORKSHOP**
**REF: DW10**

**DevWeek's first day of sessions kicks off with our keynote presentations, giving all attendees the chance to hear industry experts Kevlin Henney and Allen Holub tackle two of the biggest issues in software development.**

**DAY 1 AGENDA**

## 09.30

**KEVLIN HENNEY**

**KEYNOTE PRESENTATION**

### A SYSTEM IS NOT A TREE

Trees. Both beautiful and useful. But we're not talking about the green, oxygen-providing ones. As abstract structures we see trees all over the place – file systems, class hierarchies, ordered data structures, and so on. They are neat and tidy, nested and hierarchical – a simple way of organising things; a simple way of breaking large things down into small things.

The problem is, though, that there are many things – from modest fragments of code up to enterprise-wide IT systems – that do not comfortably fit into this way of looking at the world and organising it. Software architecture, design patterns, class decomposition, performance, unit tests... all of these cut across the strict hierarchy of trees. In this keynote, Kevlin will look at what this means for how we think and design systems, whether large or small.

## 11.30

**ROBERT SMALLSHIRE**

**UNDERSTANDING TRANSDUCERS**

Transducers – a portmanteau of "transform reducers" – are a new functional programming concept introduced into the Clojure programming language. Although transducers are actually pretty straightforward, wrapping your brain around them, especially if you're not already a competent Clojureist, can be challenging. In this session, Robert will introduce transducers by implementing them from scratch in everybody's favourite executable pseudocode: Python. We'll start with the familiar staples of functional programming and derive transducers from first principles.

**SASHA GOLDSHTEIN**

**AUTOMATING PROBLEM ANALYSIS AND TRIAGE**

What do you do when your application crashes or hangs in production? Nothing can compete with a debugger or a full process dump captured on a production system. But you can't always afford the time to analyse hundreds of crash dumps. In this session, Sasha will show you how to perform automatic dump analysis and triage using Microsoft's ClrMD, a .NET library that can explore threads, call stacks and exceptions; visualise threads and locks to form wait chains and detect deadlocks; and walk heap memory to inspect important objects for your application.

**JOHN K. PAUL**

**ECMASCRIPT 6 FOR ALL OF US**

Coming from a Java background, there was a time when JavaScript was nothing but annoyances. Now, even after we've grown to love the language, there are dozens of times when we feel the pain of missing features that Java has built in. ECMAScript 6 is changing all of that. The next version of JavaScript brings with it an amazing standard library that rivals that of Java, Python and their ilk. In this session, John will explain some of the great new additions to the language and demonstrate use cases that take advantage of ES6's elegance for client-side development.

**DROR HELPER**

**NAVIGATING THE TDD ALPHABET SOUP**

TDD, BDD, ATDD are all methodologies that enable incremental design that is suitable for Agile environments. It seems that every day a new xDD methodology is born with the promise of being better than what came before. Should you use behaviour-driven tests or plain old unit tests? Which methodology is better? And how exactly would it benefit the development life cycle? In this session, Dror will help to sort out the various methodologies – explaining where they came from, the tools they use, and discussing how and when to use each one.

**ALLEN HOLUB**

**SECURITY 101: AN INTRODUCTION TO SOFTWARE SECURITY**

As more and more of our applications move on to the web, security becomes even more critical. Good security, however, has to be built in, not tacked on as an afterthought. In this session, Allen will give you an overview of what it means to make an application secure. He'll cover topics such as security architectures, code and design review, penetration testing, risk analysis and risk-based testing, security-related requirements, static analysis, abuse cases, security operations and crypto.

**ALSO EACH DAY...**

**8.30** COFFEE & REGISTRATION | **11.00** COFFEE BREAK | **13.00** LUNCH BREAK | **15.30** COFFEE BREAK

JOIN US FOR DRINKS AT THE END OF DAY 1 – NETWORKING DRINKS SPONSORED BY amazon web services

**ALLEN HOLUB**

**KEYNOTE PRESENTATION**

## #NOESTIMATES

Estimates are always guesses – and they're always wrong. Consequently, estimate-based planning is foolhardy at best, and time spend creating them is a waste. In spite of this fact, estimates are a central part of most software-development processes, even some Agile processes. Getting rid of estimates doesn't mean that you can't plan, but you do have to go about planning in a more effective way. In this keynote, Allen will discuss both the problems surrounding an estimation culture and how to solve those problems by using actual measurements and priority-based planning.

> "A great motivational, inspirational, informative week"
>
> DIRECTOR
> 2014 DELEGATE

**KLAUS ASCHENBRENNER**

**SQL SERVER 2014 IN-MEMORY OLTP DEEP DIVE (HEKATON)**

Hekaton is the Greek word for 100 – and the goal of In-Memory OLTP in SQL Server 2014 is to improve query performance up to 100 times. In this session, Klaus will look inside the case of Hekaton and the multiversion concurrency control (MVCC) principles on which Hekaton is built. He'll start by looking at the challenges that can be solved by Hekaton, especially locking, blocking and latching within SQL Server. Based on this foundation, he'll move into the principles of MVCC, and how a storage engine and transaction manager can be built on that concept.

**DEJAN SARKA**

**INTRODUCING R AND AZURE ML**

R is a free software programming language and software environment for statistical computing, data mining and graphics. Azure Machine Learning (Azure ML) is the new Microsoft cloud service and environment for advanced data analysis, which utilises the R algorithms intensively. In this session, Dejan will introduce both R, with RStudio IDE, and Azure ML.

**ALLAN KELLY**

**DIALOGUE SHEETS FOR RETROSPECTIVES AND DISCUSSION**

Retrospectives are a key tool in the Agile toolkit, but they aren't easy. In fact, it's not just retrospectives. Teams need to learn to talk, discuss and reflect together over many things. Good conversation makes for good software. Retrospective dialogue sheets can help overcome these problems. In this hands-on session, in which everyone will get the chance to work with a dialogue sheet, discovering what one is and how to use it, Allan will discuss some of the ways teams are using them and look to the future.

**MARK SMITH**

**DESIGNING ADAPTIVE APPLICATIONS FOR THE IOS PLATFORM**

With the introduction of the iPhone 6 and 6+, we now have several form factors to consider when designing our iOS applications. In this session, Mark will focus on the designer and layout features Apple has included in iOS to help you create a single, unified storyboard that is capable of working with all variations of iOS. This will include layout constraints (auto layout), size classes, unified storyboards and the updated UISplitViewController.

**SEB ROSE**

**LESS IS MORE – AN INTRODUCTION TO LOW-FIDELITY APPROACHES**

In this session, Seb will demonstrate some key techniques that help decompose large problems. Decomposing problems is a skill all software developers need, but we're often not very good at. Whether it's stories that take longer than an iteration, or features that can't be delivered in the expected release, we've all seen the problems that tackling an over-large problem can cause. We'll work through two detailed examples to demonstrate the value of delivering small, low-fidelity pieces of work early rather than prematurely focusing on fully-polished final version.

**MICHAEL KENNEDY**

**GETTING STARTED WITH SWIFT (APPLE'S NEWEST LANGUAGE)**

Swift is Apple's newest language for building native, high performance applications for both iOS and OS X. This session will introduce you to this exciting language. Developers with a background in either C# or Python will see many similarities. Almost everyone will find Swift a much more comfortable and inviting language when compared to Objective C or C. Come and learn why it's now fun to develop (natively) for iOS and OS X.

## DAY 1 AGENDA CONTINUED

### 14.00

#### ADAM TORNHILL

**TREAT YOUR CODE AS A CRIME SCENE**

We'll never be able to understand large-scale systems from a single snapshot of the code. Instead, we need to understand how the code evolved and how the people who work on it are organised. We also need strategies that let us find design issues and uncover hidden dependencies between both code and people. Where do you find such strategies if not within the field of criminal psychology? In this session, Adam will use this approach to predict bugs, detect architectural decay and find the code that is most expensive to maintain.

#### TOBIAS KOMISCHKE

**THE PROMISED LAND (OF UX)**

In today's competitive landscape, a stellar user experience is a strong product differentiator and enabler of market success. Companies hope to get to this "Promised Land" where their own staff deploys mature UX design practices, their customers are happy, and their market share and profits increase. No one ever said that this journey to the Promised Land is easy – a lot of companies have tried and failed. In this inspirational overview session, Tobias will show the path and identify what strategic elements are critical to successful design.

#### SHAY FRIEDMAN

**ANGULARJS – THE ONE FRAMEWORK TO RULE THEM ALL**

In the last couple of years, we've seen the rise of client-side JavaScript frameworks. From almost nothing, now we have at least a dozen to choose from. One of the new kids in the block, AngularJS, comes straight from the Google offices and tries to stand out from the crowd with a complete set of tools and utilities. Some are very powerful and try to ease your way towards your SPA web site. In this session, Shay will go through the different features of AngularJS and see what makes it one of the most popular JavaScript frameworks out there.

#### RALPH DE WARGNY

**CODING THE PARALLEL FUTURE IN C++**

Many-core processors and computing platforms will be ubiquitous in the future: from multi-core and many-core CPUs to integrated GPUs to compute clusters in the cloud, it's the new parallel universe for software developers. In this session, you will learn the tools, techniques and best practices available to C/C++ developers to make sure your code is ready today to run with maximum performance and reliability in this new parallel universe.

#### DINO ESPOSITO

**ASP.NET IDENTITY, CLAIMS AND SOCIAL AUTHENTICATION**

ASP.NET Identity is the new and comprehensive membership system for the whole ASP.NET platform, including Web API and SignalR. Similar in many ways to the popular simple membership provider, ASP.NET Identity goes well beyond in a number of aspects: replaceable storage, flexible representation of user profiles, external logins, claims-based authentication and role providers. Dino will use the Identity API to set up social authentication via Facebook and Twitter, and to collect any user information made available by social networks.

### 16.00

#### ED COURTENAY

**INVERSION OF CONTROL 101**

The dependency injection/inversion of control design pattern is an important technique that helps to write testable and maintainable code. In this session, Ed will debunk the myth that it's hard to understand or only for "enterprise development", and demonstrate how to use it in everyday code. This demo-led session will discuss the rationale behind dependency injection, demonstrate injection with and without a dependency container, as well as writing a simple container from scratch.

#### PEARL CHEN

**INTERNETTING YOUR THINGS**

From big consumer success stories, such as the Nest Thermostat, to more offbeat monitoring systems, such as Botanicalls (which lets your plant call you when it is thirsty), your *things* are finding their own voice through small but powerful embedded microcontrollers. The market for the Internet of Things (IoT) and wearables is exploding. But what are your options for getting started with making physical things when you're more used to writing software? In this session, Pearl will go through a few hardware options that are great for hobbyists and rapid prototyping.

#### PHIL LEGGETTER

**PATTERNS AND PRACTICES FOR BUILDING ENTERPRISE SCALE HTML5 APPS**

Developing large apps is difficult. Ensuring that code is consistent, maintainable, testable and has an architecture that enables change is essential. When it comes to large server-focused apps, solutions to some of these challenges have been tried and tested. But, how do you achieve this when building HTML5 single-page apps? In this session, Phil will highlight signs to watch out for as your HTML5 SPA grows, and patterns and practices to help you avoid problems. He will also explain the architecture that their HTML5 apps have that is core to ensuring they scale.

#### OREN RUBIN

**TEST AUTOMATION DONE RIGHT: THE HOLY GRAIL OF CONTINUOUS DEPLOYMENT**

In this session, Oren will explain all there is to know about end-to-end test automation. Starting with the basics and a comparison to unit testing, he will then drill down into what is considered uncharted territory for many developers. You will learn the best practises and design patterns, common pitfalls, and – most importantly – the full ecosystem and how it connects to your existing toolchain. You will learn about different approaches to UI verifications, and see real industry use cases and bugs.

#### SASHA GOLDSHTEIN

**WHAT'S NEW IN VISUAL STUDIO 2015?**

In this session, Sasha will lead us through a sample of the new Visual Studio 2015 features, ranging from developer productivity to low-level C++ code optimisations. In a series of quick-paced demos, we will show how Visual Studio 2015 makes key diagnostics experiences easier, improves IntelliTrace analysis, helps developer collaboration and productivity, and produces higher-quality and faster code for both managed (with .NET Native) and C++ applications. If you're using Visual Studio, you can't afford to miss this talk!

**ALSO EACH DAY...**

**8.30** COFFEE & REGISTRATION | **11.00** COFFEE BREAK | **13.00** LUNCH BREAK | **15.30** COFFEE BREAK

JOIN US FOR DRINKS AT THE END OF DAY 1 – NETWORKING DRINKS SPONSORED BY amazon web services

### MICHAEL KENNEDY

**SCALING THE NOSQL WAY WITH MONGODB**

The great promise of the NoSQL databases has been their ability to scale out rather than scaling up. In this session, Michael will look at a concrete example of scaling one of the most generally useful and most widely deployed NoSQL database: MongoDB. He'll explore why you might need to scale out, and you'll see the full spectrum of choices for scaling (replication, sharding, geo-replication, etc). NoSQL document databases typically outperform RDBMSes on single servers but with the ability to scale out they can truly achieve an entirely new level of performance.

### IQBAL KHAN

**LEARN HOW TO SCALE .NET APPS IN MICROSOFT AZURE WITH DISTRIBUTED CACHING**

Discover the scalability bottlenecks for your .NET applications in Microsoft Azure, and how you can improve their scalability with distributed caching. This session provides a quick overview of scalability bottlenecks, and answers some key questions: What is distributed caching and why is it the answer in Microsoft Azure? Where in your application can you use distributed caching? What are some important features in a distributed cache? You'll also see hands-on examples of using a distributed cache.

### NEAL FORD

**BUILDING MICROSERVICE ARCHITECTURES**

Inspired by the success of companies such as Amazon and Netflix, many organisations are moving rapidly towards microservice architectures. This style of architecture is important because it's the first architecture to fully embrace the Continuous Delivery and DevOps revolutions. In this session, Neal will cover the motivations for building a microservice architecture, some considerations you must make before starting (such as transactions versus eventual consistency), how to determine service partition boundaries, and ten tips for success.

### NUNO GODINHO

**EVENT HUBS, AZURE STREAMING AND AZURE ISS**

Recently, there has been a lot of talk around IoT, M2M, big data and similar topics. It's important to understand how we can take advantage of these concepts, and how they can help us achieve our goals. Fortunately, Microsoft has some solutions for us. These are Azure Event Hubs, Azure Stream Analytics and Azure Intelligence Systems Service (ISS). In this session, Nuno will explore these three topics, demonstrate their interconnectivity, and show how they provide the perfect answer for our next-generation solutions and interactions.

### JULES MAY

**IF CONSIDERED HARMFUL: HOW TO ERADICATE 95% OF ALL YOUR BUGS IN ONE SIMPLE STEP**

In 1968, CACM published a letter from Edgar Dijkstra, called "Go To statement considered harmful". In it, he explained why most bugs in programs were caused by Gotos, and appealed for Goto to be expunged from programming languages. But Goto has a twin brother, which is responsible for nearly every bug that appears in our programs today. That twin is If. In this session, Jules revisits Dijkstra's original explanation to show why If and Goto have the same pathology, and how you can avoid it.

### SAHIL MALIK

**TOP 10 JAVASCRIPT TIPS**

JavaScript, the *lingua franca* of the web, is incredibly freeform and therefore hard to get right. We've all hacked JavaScript, but what do you need to know when you are doing big and complex JavaScript projects? This isn't your Dad's browser, y'know! So you have written JavaScript, but want to go beyond the basics? In this session, Sahil will show you the JavaScript concepts that every modern JavaScript developer needs to know.

### MIKE WOOD

**MESSAGING PATTERNS**

There are many reasons why asynchronous messaging should be introduced in applications, as well as many approaches in incorporating messaging subsystems. In some cases, intensive workloads need to be pushed to back-end processing, or perhaps specialised (and often expensive) resources need to be utilised to perform certain operations. In this session, Mike will cover several scenarios where introducing messaging can help, discuss a few messaging patterns, and look at abstracting your messaging subsystem to guard against evolving technology and designs.

### GARY SHORT

**DATA SCIENCE FOR FUN AND PROFIT**

Make no mistake: data science can be hard, but it can also be fun. In this session, Gary will introduce you to classic and Bayesian statistics and machine learning, all through the medium of predicting horse-racing results. He'll explore a number of techniques for making such predictions and finish by combining them into a powerful "mixed model" prediction engine that's sure to pick the next big winner. This session won't only improve your knowledge, it'll improve your bank balance too! (Note: Session may not improve bank balance.)

### AUSTIN BINGHAM

**HIGH-QUALITY DECISION MAKING WITH OPEN DESIGN PROPOSALS**

Making complex decisions in software design involves balancing many factors, and maintaining that balance can be challenging. By opening up the decision process for evolution, we can harness the insight of fellow developers, communicate plans and designs more effectively, and produce a useful record of the work we do. In this session, Austin will look at a specific technique: Open Design Proposals. He'll examine implementations of this approach, see why it's effective, and show how development teams can use it to manage their own decision making.

### JAMES MONTEMAGNO

**IBEACONS AND CONTEXTUAL LOCATION AWARENESS IN IOS AND ANDROID APPS**

iBeacons are taking the world by storm – from retail stores to major sporting events, you'll soon be finding iBeacons just about everywhere. This gives you the ability to enable any number of device proximity-based scenarios that were never before possible. In this session, James will explain what an iBeacon is, how they work, how you would want to use them, and how to get started making apps in both Android and iOS. All demonstrations will be coded in C#, but will be applicable to any iOS or Android developer in any language.

### KEVLIN HENNEY

**FP 4 OOP FTW!**

Although not yet fully mainstream, functional programming has finally reached a critical mass of awareness among developers. The problem, however, is that many developers are up against an even greater critical mass of existing code. Much of this code purports to be object oriented, but beyond the use of the class keyword, falls somewhat short of putting the OO to good use. Many techniques identified as functional have broader applicability. In this session, Kevlin will explore how some FP habits can be incorporated into OOP practice.

### SAHIL MALIK

**TOP 10 JAVASCRIPT DEBUGGING TRICKS**

We've all been writing lots of JavaScript code lately. But JavaScript is incredibly free form, and that sharp double-edged sword can also make debugging JavaScript errors a lot more difficult. The new operating system is the browser, and complex JavaScript pages cannot ignore performance, or their unpredictable behaviour under different bandwidths. In this session, Sahil will show you some really useful debugging techniques and demonstrate how to use each browser for its best capabilities. Ninety minutes spent here will save you hours in your day job.

**Day 2 of DevWeek continues with a packed programme of workshops and breakout sessions – and don't forget that if you can't make it to any of the talks you're interested in, as a registered delegate you'll be able to catch up later online at the DevWeek website.**

## DAY 2 AGENDA

### 09.30

#### ADAM TORNHILL

**CODE THAT FITS YOUR BRAIN**

People think, remember and reason in a very different way from that in which code is presented. So how should code look to make it both easier to understand and maintain? To see what really works, we need to look across languages and paradigms. In this session, Adam will start with common problematic constructs such as null references, surprising corner-cases and repetitive code, and discuss the cognitive costs and consequences of each. He'll then apply ideas from object-orientation, functional programming and lesser-known array languages to explore better approaches.

#### TOBIAS KOMISCHKE

**VISUAL DESIGN FOR NON-DESIGNERS: IT'S NOT JUST ABOUT COLOUR!**

In an ideal world, front-end developers don't need to worry about visual design because they get specs and assets from professionally trained designers. The reality is that developers often need to make their own decisions about how to make the UI attractive. In this session, Tobias will provide a solid base knowledge about what constitutes attractiveness and what design principles can be applied to boost the visual appeal of UIs. By the time you leave, you'll be ready to step up to the mark when designers are nowhere to be found!

#### PHIL LEGGETTER

**WHY YOU SHOULD BE USING WEB COMPONENTS – AND HOW**

Web Components are touted as the future of web development. In this session, Phil will explain what Web Components are, the state of native support in web browsers, what your options are for building componentised web apps right now using AngularJS, Ember, Knockout or React, and why Web Components probably *are* the future of web development. He'll also cover the benefits of a component-based architecture and how it helps when building JavaScript apps, as well as how components can communicate in a loosely coupled way, and why.

#### ED COURTNEY

**AN INTRODUCTION TO TYPESCRIPT**

JavaScript is the scripting glue that holds the web together – largely because of its flexibility. This flexibility also means that it can be difficult to manage, especially in large-scale applications. In this session, using real-world examples, Ed will explore some of the problems with client-side JavaScript development as a motivating example and introduce TypeScript as a way of solving some of these issues. He'll also explore how existing JavaScript codebases can be targeted by TypeScript and how it can be integrated into build systems.

#### SASHA GOLDSHTEIN

**COOL LIBRARIES FOR MODERN C++**

The C++ standard library dates back to the 1990s, but that doesn't mean there aren't new and exciting frameworks to use in your C++ application. In this session, Sasha will look at some brand-new and some existing C++ libraries that can speed up C++ cross-platform development rapidly. Some of the libraries we might cover include: Casablanca (C++ REST SDK), Cinder (creative coding), Boost (general-purpose), Google Test (unit testing), SOCI (modern database access) and many others. There's something for everyone!

### 11.30

#### PAVEL SKRIBTSOV

**COMPUTING LIKE THE BRAIN: AN INTRODUCTORY GUIDE TO AI**

Every now and again, every professional developer faces a program that he or she has trouble writing. Try to imagine an algorithm that has to differentiate a dog from a cat. They come in different shapes and sizes, and there is no single feature that could discriminate between the two. Any attempt to code that algorithm manually using deep-nested "if/else" branches is doomed. Human beings, on the other hand, have no trouble with this task. In this session, Pavel will introduce the basics of an artificial intelligence-based approach to solving these problems.

#### JOE NATOLI

**THINK FIRST: WHY GREAT UX STARTS BETWEEN YOUR EARS (AND NOT ON THE SCREEN)**

When developers are tasked with improving UX, their focus tends to be on the screen: elements, interactions, workflow, often accompanied by the worrying cry, "I'm not a UI designer!" Fortunately, Joe has good news: you can still design great user experiences without a shred of visual design talent. In this session, Joe will show you how changing the way you think about features, functions and implementations can make a massive, positive change in the experience people have with your UI and your product.

#### ALLEN HOLUB

**KNOCKOUT: AN INTRODUCTION**

The Knockout framework is a standalone implementation of the MVVM (Model-View-ViewModel) pattern, which is one of the best user-interface architectures for web applications. It provides an alternative to AngularJS – more limited in scope but smaller and, in some contexts, faster. In this session, Allen will look at Knockout's architecture and how to leverage that architecture to build highly interactive web-application user interfaces. The session will include several code (JavaScript) examples.

#### DROR HELPER

**BATTLE OF THE .NET MOCKING FRAMEWORKS**

Writing unit tests is hard, isn't it? You need an entire set of tools just to start. One of the crucial decisions when building this set is picking up a mocking framework. But beware – the mocking framework you choose has the ability to make or break you! In this session, Dror – at one time a mocking framework developer – will cover the capabilities and functionality of the leading frameworks, showing the good and the bad of the different options (both free and commercial), and making them battle to the death!

#### SHAI REZNIK

**BUILD PRODUCTION-READY JAVASCRIPT APPS WITH GRUNT**

In this session, Shai will deliver an overview of the steps required in order to build JavaScript apps and get them ready for deployment. He'll cover build theory, asking "Why build in JS?" He'll also talk about the build steps and then jump to Grunt, explaining what it is and providing a live demo. Finally, Shai will cover the scaffolding tool, Yeoman. This session is intended to be both funny and informative, so get ready to have a good time while picking up some essential tips to make your day job that much easier.

**ALSO EACH DAY...**

**8.30** COFFEE & REGISTRATION | **11.00** COFFEE BREAK | **13.00** LUNCH BREAK | **15.30** COFFEE BREAK

**KLAUS ASCHENBRENNER**

**THE DANGEROUS BEAUTY OF BOOKMARK LOOKUPS**

You know Bookmark Lookups in SQL Server? You like their flexibility to retrieve data? If you do, then you should be warned that you are dealing with the most dangerous concept in SQL Server! Bookmark Lookups can lead to massive performance losses that will devastate your CPU and I/O resources! In this session, Klaus will provide a basic understanding of Bookmark Lookups and how they are used by SQL Server. After laying out the foundations, he'll talk in more detail about the various performance problems they can introduce.

**DEJAN SARKA**

**DATA MINING ALGORITHMS WITH SQL SERVER AND R (PART 1)**

Data mining is gaining popularity as the most advanced data analysis technique. With modern data mining engines, products and packages, such as SQL Server Analysis Services (SSAS) and R, data mining has become a black box. It's possible to use data mining without knowing how it works, but this can lead to many problems, such as using the wrong algorithm for a task, misinterpretation of the results and more. In this session (and Part 2, at 11.30), Dejan will explain how the most popular data mining algorithms work and when to use each one.

**NEAL FORD**

**CONTINUOUS DELIVERY FOR ACHITECTS**

Yesterday's best practice is tomorrow's anti-pattern. Architecture doesn't exist in a vacuum: a painful lesson developers who built logically sound but operationally cumbersome architectures learned. Continuous Delivery is a process for automating the production-readiness of your application every time a change occurs to code, infrastructure or configuration. In this session, Neal will take a deep dive into the intersection of the architect role and the engineering practices in Continuous Delivery.

**JAMES MONTEMAGNO**

**IOS AND ANDROID DEVELOPMENT FOR C# DEVELOPER WITH XAMARIN**

As the mobile landscape continues to expand and evolve, managing multiple codebases in different programming languages and development tools can quickly become a nightmare. Wouldn't you love to build native UIs for iOS, Android and Windows Phone from a single codebase? In this session, James will show how to leverage the awesome features of C# and combine them with Xamarin technology to create beautiful, native, cross-platform, mobile apps from a shared C# codebase, with the tools that you love.

**SAHIL MALIK**

**LEARN ANGULARJS: THE ROAD TO POWERFUL, MAINTAINABLE APPLICATIONS**

JavaScript, by its nature, makes it difficult to write maintainable code. HTML, by its nature, is loosely structured. AngularJS fixes both of those. It's a structural framework for dynamic web apps, allowing you to extend HTML's syntax, enabling you to write powerful, maintainable applications succinctly.

In this workshop, Sahil will build on your existing knowledge of JavaScript and teach you the ins and outs of AngularJS. There are plenty of examples, which will walk you through a basic introduction, models, controllers and views in Angular, templates and databinding, services and dependency injection, directives, routing and single-page applications.

*For a full description of the workshop, please see Page 21*

**ONE-DAY WORKSHOP**

**WORKSHOP REF: MC01**

**ANDY CLYMER & RICHARD BLEWETT**

**SOLID ASYNC PROGRAMMING IN .NET**

In this special two-day workshop, Andrew and Richard will take you through the core skills required to successfully develop async and multithreaded code, both in the .NET and web worlds. Not only do we cover the core APIs, but also how they are used effectively, tested and debugged.

*For a full description of the workshop, please see Page 20*

**TWO-DAY WORKSHOP**

**WORKSHOP REF: MC03**

**MICHAEL KENNEDY**

**APPLIED NOSQL WITH MONGODB AND PYTHON**

NoSQL is a hot topic in the tech industry today. But what exactly *is* NoSQL and should I use it to build my next application? In this session, Michael will dig into why NoSQL databases are sweeping the industry and discuss the trade-offs between the various types (key-value stores vs document databases, for example). He will explore the most broadly applicable variant of NoSQL, document databases, through hands-on demos with the most popular and successful of the document databases, MongoDB.

**DEJAN SARKA**

**DATA MINING ALGORITHMS WITH SQL SERVER AND R (PART 2)**

Data mining is gaining popularity as the most advanced data analysis technique. With modern data mining engines, products and packages, such as SQL Server Analysis Services (SSAS) and R, data mining has become a black box. It is possible to use data mining without knowing how it works, but this can lead to many problems, such as using the wrong algorithm for a task, misinterpretation of the results and more. In this session (following on from Part 1, at 09:30), Dejan will explain how the most popular data mining algorithms work and when to use each one.

**DINO ESPOSITO**

**DDD MISCONCEPTIONS**

For too long, domain-driven design (DDD) has been sold as the ideal solution for very complex problems that only a few teams are actually facing. While technically correct, this statement sparked a number of misconceptions. In fact, DDD is only an approach to the design of software systems and is driven by the domain of the problem. In this session, Dino will clear the ground around DDD, emphasising the theoretical pillars of the approach: ubiquitous language and bounded context.

**MIKE WOOD**

**5 LIGHTWEIGHT MICROSOFT AZURE FEATURES FOR FAST-MOVING MOBILE DEVS**

Mobile development has exploded, and everyone has an idea they want to try out. But bootstrapping a mobile app doesn't always seem that easy. Consumers demand slick user experiences and the ability to share data across a plethora of devices and platforms, while we're trying to get a minimal viable product out the door to test our ideas as fast as possible. Thankfully, Azure has powerful features available to help. In this session, Mike will take a practical look at five features of Azure that are useful for mobile developers of any platform.

## DAY 2 AGENDA CONTINUED

### 14.00

**ED COURTENAY**

**BEHAVING LIKE A GIT, AND GETTING AWAY WITH IT**

Although Git has rapidly become almost a *de facto* standard in recent years, it can be intimidating or confusing for those transitioning from other systems or those new to using source control. In this session, Ed will explain how to use Git effectively, how to navigate your way around a repository, and how to work as part of a team. He'll attempt to cut through the mystique and demonstrate how easy it can actually be to use. Then he'll go on to show some of the more advanced ways of working with Git.

**HOWARD DEINER**

**CONTINUOUS DELIVERY: THE WHYS, WHATS, AND HOWS**

DevOps is commonly believed to be accomplished by having the development staff collaborate more closely with the operations staff. That's definitely necessary, but woefully inadequate to achieve the goal of faster and better delivery in the "last mile" of an Agile shop. In this session, Howard will discuss the rationale behind Continuous Delivery, along with specific practices to get you started on making your sprints toward customer satisfaction less tiring and more enjoyable for everyone involved.

**SASHA GOLDSHTEIN**

**MAKING .NET APPLICATIONS FASTER**

Speed is king on mobile devices, embedded systems, and even run-of-the-mill desktop applications that need to start up quickly and deliver good performance on low-power machines. In this session, Sasha will review a collection of practical tips you can use today to make your .NET applications faster. He'll talk about choosing the right collection, improving start-up times, reducing memory pressure, and many other techniques for quickly improving your app's performance.

**MICHAEL HABERMAN**

**UNIT TESTING AND E2E TESTING USING JS-BASED FRAMEWORKS**

Unit testing and end-to-end (e2e) testing are the tools to enforce stability on applications. They create an environment that ensures our code does what it was designed to do. Recently, web application developers are looking to identify the best testing option, as their applications are getting increasingly large and more complex. In this session, Michael will review two methods for testing web applications in different JS-based frameworks: the unit-testing approach and end-to-end testing. He will also review the benefit of combining the two.

**ANTHONY SNEED**

**SECURING WEB APIs THE NEW WAY WITH OWIN AND KATANA**

Sometimes the technology landscape is changing so fast, it feels like you're standing on quicksand. That is certainly the case with ASP.NET Web API, the new OWIN hosting model and Microsoft's Katana implementation. In this session, Anthony will show how to correctly apply security at the transport level to ensure confidentiality, integrity and server authentication, as well as the nuts and bolts of configuring SSL for both web and self-hosted web APIs using the new OWIN hosting model.

### 16.00

**KEVLIN HENNEY**

**PROGRAMMING WITH GUTS**

These days, testing is considered a sexy topic for programmers. Who'd have thought it? But what makes for good unit tests (GUTs)? There's more to effective unit testing than just knowing the assertion syntax of a testing framework. Testing represents a form of communication and, as such, it offers multiple levels and forms of feedback, not just basic defect detection. Effective unit testing requires an understanding of what forms of feedback and communication are offered by tests. In this session, Kevlin will explore exactly what makes a good unit test.

**JULES MAY**

**TEAM BUILDING**

No programmer is an island. Modern programs are created by teams of developers. And everybody knows you need great teams to build great products – so you need to build your teams carefully. But what, exactly, makes a great programming team? Great programming skills? Great interpersonal skills? Working-all-night-because-the-boss-has-thrown-a-fit skills? Turns out, it's none of these. In this session, Jules will reveal that what makes a programming team great is exactly the same stuff that makes any other team great – and most programming teams don't have it.

**IDO FLATOW**

**ASP.NET VNEXT: REIMAGINING WEB APPLICATION DEVELOPMENT IN .NET**

ASP.NET vNext is being designed from the bottom up to be a lean and composable .NET stack for building web and cloud-based applications. Envision an ASP.NET stack where MVC, Web API, and web pages are all merged into the same framework, where you have a server-optimised version of ASP.NET with a smaller memory footprint. This is the new ASP.NET vNext. In this session, Ido will explore the ecosystem of ASP.NET vNext, its new project system and configuration system, and how to use it to build exciting web applications.

**SHAY FRIEDMAN**

**CHROME DEVELOPER TOOLS – A DEEP DIVE**

Every developer needs a set of tools, especially web developers that bend under the pressure of multiple languages, environments, IDEs and what not. One of the most comprehensive toolsets out there today is Chrome Developer Tools. It contains so many amazing features beyond the common ones, and it's just a shame most developers don't know about them! In this session, Shay will tell you all about the known and less-known features of Chrome Developer Tools, and you'll see how your everyday web development can become easier with just a few simple steps.

**SASHA GOLDSHTEIN**

**PRACTICAL C# 6 AND BEYOND**

Visual Studio 2015, .NET 2015 and C# 6 are just around the corner. The new language features have been out of the bag for a while now, but how do you apply them effectively? How do you refactor existing code to be shorter and sweeter? In this fast-paced session, Sasha will lead us through experiments with the new language features, including expression-bodied members, enhancements to automatic properties, null propagation, string interpolation and many others.

**ALSO EACH DAY...**

**8.30** COFFEE & REGISTRATION | **11.00** COFFEE BREAK | **13.00** LUNCH BREAK | **15.30** COFFEE BREAK

### ALLEN HOLUB

**ZEROMQ AND RABBITMQ: MESSAGING FOR AGILITY AND SCALABILITY**

Messaging is an essential technology in high-volume, dynamically scalable server applications. It's the most effective way to pass non-time-critical information between servers, and to distribute work within a server farm. At the inter-server level, messaging is ideal for use with remote databases, monitoring, logging and so on, and a far better solution to intra-server data sharing than a shared database. Allen looks at messaging from an architectural perspective, with practical examples using RabbitMQ and ZeroMQ.

### GARY SHORT

**HADOOP KICKSTARTER FOR MICROSOFT DEVS**

Big data is the new shiny thing right now, and if you read the blogosphere you'd be forgiven for thinking it was a tool just for Linux devs – or worse, only for those annoying hipsters with their shiny Macs. Nothing could be further from the truth. Windows makes an excellent platform for Hadoop and, in this session, Gary will show you everything you need to know to get started. From downloading and installing, to writing your first map-reduce job, using both the streaming API and the SDK. This session will cover it all, so come along and join the big data wave!

### SANDER HOOGENDOORN

**INDIVIDUALS AND INTERACTIONS OVER PROCE$$E$ AND FOOLS**

The first statement in the Agile Manifesto favours individuals, teams, interaction and collaboration over processes and tools. But there are two sides to every story. When it comes to tools, the Agile Manifesto is often misinterpreted, in the sense that it's wrong to use tooling in Agile projects. Despite this, more and more vendors are trying to jump on the Agile bandwagon and sell their tools as being the most Agile toolset available. In this session, Sander shines a critical light on the sense and nonsense of tools in the Agile field.

### MARK SMITH

**GETTING YOUR MOBILE APPS READY FOR THE WORLD**

Localising your applications can open up a whole new audience of users for your software. In this session, Mark will take a look at how to get your application ready for localisation and how to then utilise the built-in services of iOS, Android and Windows Phone to display proper information for different cultures and regions.

### SAHIL MALIK

**LEARN ANGULARJS: THE ROAD TO POWERFUL, MAINTAINABLE APPLICATIONS**

One-day workshop continues from the morning session.

*For a full description of the workshop, please see Page 21*

**ONE-DAY WORKSHOP**

**WORKSHOP REF: MC01**

### ANDY CLYMER & RICHARD BLEWETT

**SOLID ASYNC PROGRAMMING IN .NET**

Two-day workshop continues from the morning session.

*For a full description of the workshop, please see Page 20*

**TWO-DAY WORKSHOP**

**WORKSHOP REF: MC03**

### KLAUS ASCHENBRENNER

**UNIQUEIDENTIFIERS AS PRIMARY KEYS IN SQL SERVER**

Is it good practice to use uniqueidentifiers as primary keys in SQL Server? They have a lot of pros for devs, but DBAs just cry when they see them enforced by default as unique clustered indexes.
In this session, Klaus will cover the basics of uniqueidentifiers: why they are sometimes bad and sometimes good; and how to discover if they affect the performance of your performance-critical database.
If they are having a negative impact, you will also learn some best practices you can use to resolve those limitations without changing your underlying application.

### DAN CLARK

**AUTOMATING SSIS PACKAGE CREATION WITH BIML**

(BIML) is a powerful XML-based markup language that allows you to generate SSIS packages programmatically. Using BIML along with C#, you can create metadata-driven packages, greatly reducing development time and increasing consistency across the team. In this session, Dan will show you how to automate your SSIS package creation using the power of BIML and C#. You'll see how to create a template for loading dimension tables that will greatly increase your productivity.

### SEB ROSE

**CUCUMBER AND SPECFLOW AS PART OF YOUR DEVELOPMENT PROCESS**

Behaviour-driven development (BDD) and specification by example (SBE) are quite recent additions to the software development toolbox. Sometimes it feels like we're using a hammer to drive in a screw.
So, in this session, Seb will explore what they're good for and when to use them. He'll also look at what problems they don't help with and when not to use them. By the end of this session, you'll know enough to decide whether your problems are more like a screw or a nail – and whether Cucumber/SpecFlow is the right hammer.

### NUNO GODINHO

**IOT & M2M: HOW ARE THEY CHANGING THE WORLD WE LIVE IN?**

Internet of Things (IoT) is here, and every day a new sensor or device starts to generate more data. With that, more and more machine-to-machine (M2M) communications start to happen, which make our solutions behave differently and face new issues. In this session, Nuno will look at how both of these new "buzz words" are changing the world we live in, from fitbit to Google Glass and smart watches. How can we prepare for this? How can we anticipate and get some business opportunities from it? Join us and find out.

## DevWeek Day 3 sees the concluding part of our two-day workshop on asynchronous programming in .NET, plus a new workshop on Entity framework, plus 36 more breakout sessions covering a wide range of topics.

### DAY 3 AGENDA

**09.30**

**MICHAEL KENNEDY**

**PYTHON: AN AMAZING SECOND LANGUAGE FOR .NET DEVELOPERS**

The modern software development landscape is a terrain of many platforms and technologies. Gone are the days where knowing one technology really well was enough to stay on the cutting edge. Even as we know we should learn more and branch out, that choice is increasingly difficult as the technology options explode. In this session, Michael offers one very solid choice: Python. It may seem like a very different language and ecosystem from .NET but beneath the surface, there are many more similarities than differences.

**DINO ESPOSITO**

**THE (DRAMATIC?) IMPACT OF UX ON SOFTWARE ARCHITECTURE AND DESIGN**

Always neglected in favour of domain analysis and modelling, the presentation layer of applications receives little attention. But whether your application is web, mobile or desktop, the presentation layer is the face it shows to users. Dino will discuss a design approach that starts from requirements and builds the system from top to bottom, focusing on use-cases, screens and overall user experience measured by a new professional figure – the UX architect – and backed by new, but partially green, tools such as UXPin and Balsamiq.

**CHRISTOS MATSKAS**

**MEET THE NEW KID ON THE BLOCK: MICROSOFT ASP.NET 5**

Imagine if you could write an ASP.NET application using your favourite text editor, compile it and run it on Mac OS X. Imagine if you could mix and match Web Forms, MVC, Web API and SignalR within a single project. How would it feel to create a faster, leaner and more memory-efficient ASP.NET application that has been freed from the shackles of Windows, and all you need are your coding skills, a couple of NuGet packages and your imagination? In this session, Christos provides an intro to Microsoft's ASP.NET 5 – the "new kid on the block".

**DROR HELPER**

**UNIT TESTING PATTERNS FOR CONCURRENT CODE**

Getting started with unit testing is not hard, the only problem is that most programs are more than a simple calculator with two parameters and a return value that's easy to verify. Writing unit tests for multi-threaded code is harder still. In this session, Dror will demonstrate useful patterns that he has discovered over the years, and that have helped him to test multi-threaded and asynchronous code and enabled the creation of deterministic, simple and robust unit tests. He'll also point out the pitfalls to avoid.

**EOIN WOODS**

**SYSTEM SECURITY BEYOND THE LIBRARIES**

Security is now important to all of us, not just people who work at Facebook. But it's a complicated domain, with a lot of concepts to understand. In any technical ecosystem, there is a blizzard of security technology, as well as generic concepts such as keys, roles, certificates, trust, signing and so on. Yet none of this is useful unless we know what problem we're really trying to solve. In this session, Eoin will dive into the fundamentals of system security to introduce the topics we need to understand in order to decide how to secure our systems.

**11.30**

**PAVEL SKRIBTSOV**

**DEEP LEARNING: THE HANDCRAFTED CODE KILLER**

In this session, Pavel will reveal how a new direction in artificial intelligence, called "deep learning", is gradually reducing demand for hand-crafted code for intellectual data analysis, primarily in the area of feature extraction. He will explain why the internet giants (Google, Microsoft etc) are interested in deep learning, and the connection with big data projects. He will also cover practical examples of applying existing deep-learning software frameworks to an image-recognition problem.

**AMY CHENG**

**DRAWING WITH JAVASCRIPT: ANIMATIONS AND INFOGRAPHICS**

A number of libraries and frameworks allow developers to use JavaScript to create engaging visuals without switching programming languages. So let's explore the visual (and fun) side of JavaScript! In this session, Amy will provide a whirlwind tour of a few libraries and frameworks that let you create animations, simple drawings and infographics with JavaScript. First, she'll examine the advantages (and disadvantages) of using JavaScript for graphics and animations. Then she'll go through "Hello World" examples of a few JavaScript-based libraries.

**SANDER HOOGENDOORN**

**INTRODUCING AND EXTENDING BOOTSTRAP**

Bootstrap is by far the most popular web framework of all, with many ready-to-use styles and components in CSS and JavaScript. In this session, Sander will show you how to build a basic web site, leveraging the many components of the Bootstrap framework. He will then go on to show the use of additional frameworks and libraries to add drop-down support, icons and date pickers to your web pages, and how to build additional reusable components using Razor syntax, in JSF, and applying Angular directives. Of course, Sander's talk will be illustrated with many coding demos.

**ALLEN HOLUB**

**DBC (DESIGN BY CODING)**

Design by Coding (DbC) is a way to develop an architecture incrementally as you code. It builds on test- and behaviour-driven-development techniques, but adds a focus on the "story" that's central to all Agile processes. The process answers the question of how you can build a coherent Agile system incrementally, without a formal up-front design process. In this session, Allen will explain how DbC eliminates the need for a separate design phase in the development process, since your code is effectively your design artefact.

**TONI PETRINA**

**AWESOME THINGS YOU CAN DO WITH ROSLYN**

Roslyn, the revamped compiler for C# and Visual Basic.NET, goes beyond a mere black-box compiler and gives us limitless possibilities. Besides enabling a new era for C# as a language, it gives everyone a chance to utilise compiler powers for building custom tools. It acts as a CaaS, or Compiler as a Service, which allows you to plug in at any point in the compilation process. But what can you do with it? In this session, Toni will show you, demonstrating how you can build Visual Studio extensions, create your own editors or host C# compiler to form a scripting environment.

**ALSO EACH DAY...**

**8.30** COFFEE & REGISTRATION | **11.00** COFFEE BREAK | **13.00** LUNCH BREAK | **15.30** COFFEE BREAK

### KLAUS ASCHENBRENNER

**HEADACHE GUARANTEED: DEADLOCKING IN SQL SERVER!**

SQL Server needs its locking mechanism to provide the isolation aspect of transactions. As a side-effect, your workload can run into deadlock situations – a guaranteed headache for any DBA! In this session, Klaus will look into the basics of locking and blocking in SQL Server. Based on that knowledge, you will learn about the various kinds of deadlocks that can occur in SQL Server, how to troubleshooting them, and how you can resolve them by changing your queries, your indexing strategy and your database settings.

### DEJAN SARKA

**DATA EXTRACTION AND TRANSFORMATION WITH POWER QUERY AND M**

Power Query, a free add-in for Excel 2010 and 2013 and part of the Power BI suite in Excel 365, is a powerful tool. Dejan will show how you can use Power Query to gather all kinds of data, from databases to web sites and social media, inside Excel data models. In this way, you can make Excel an analysing engine for structured and unstructured data. In addition to the queries you can create through the UI, there is a fully functional language, called M, behind the scenes. This session introduces both Power Query and M.

### ALLEN HOLUB

**MICRO SERVICES: A CASE STUDY**

In this session, Allen will take a deep dive into a micro-service implementation. He'll look at both the architecture and the implementation of authentication and comment-management micro-services suitable for use in a blog or similar application. The core system is written in Java, so you'll need to know Java, C++, C#, or equivalent to follow along easily. Auxiliary technologies include Mongo, JavaScript, AngularJS and Bootstrap, so this session provides a real-world example of how those technologies work. You don't need to be familiar with any of them, though.

### SASHA GOLDSHTEIN

**MODERNISING C++ CODE**

The C++ standard library dates back to the 1990s, but that doesn't mean there aren't new and exciting frameworks to use in your C++ application. In this session, Sasha will look at some brand-new and some existing C++ libraries that can speed up C++ cross-platform development rapidly. Some of the libraries we might cover include: Casablanca (C++ REST SDK), Cinder (creative coding), Boost (general-purpose), Google Test (unit testing), SOCI (modern database access) and many others. There's something for everyone!

### ANTHONY SNEED

**SOUP TO NUTS: DEVELOPING REAL-WORLD BUSINESS APPS WITH ENTITY FRAMEWORK AND ASP.NET WEB API**

Performance. Scalability. Maintainability. Testability. Security. Today's application developers need to build systems that are designed to achieve these goals from the outset. In this in-depth workshop, Anthony will take you beyond the basics, to learn how to build RESTful services that are robust, scalable and loosely coupled, using dependency injection with repository and unit of work design patterns.

But there's more to building loosely coupled systems than applying a set of design patterns. Anthony will show you how to harness the power of code generation by customising T4 templates for reverse engineering Code First classes from an existing database, in order to produce entities with persistence concerns that are completely stripped away. You'll also learn ninja techniques for handling cyclical references with code-based configuration and using efficient binary formatters, all without polluting your entities with mapping, serialisation or validation attributes.

This workshop will focus on developing real-world business apps using the Entity framework and ASP.NET Web API.

**ONE-DAY WORKSHOP**

**WORKSHOP REF: MC02**

### ANDY CLYMER & RICHARD BLEWETT

**SOLID ASYNC PROGRAMMING IN .NET**

Two-day workshop continues from the previous day's session.

*For a full description of the workshop, please see Page 20*

**TWO-DAY WORKSHOP**

**WORKSHOP REF: MC03**

### SASHA GOLDSHTEIN

**RAVENDB: THE .NET NOSQL DATABASE**

The big data hype is all about NoSQL databases that can support huge amounts of data, replication, scaling and super-fast queries. RavenDB is the best NoSQL database for .NET developers, because it has a first-class .NET client with a LINQ API. In this session, Sasha will show you how to model data as documents for storage in RavenDB; how to query the data efficiently; and how to construct indexes that will help you get the data you need in just a few milliseconds. We'll also review full-text search and query suggestions support, which make it very easy to add search capabilities.

### DAN CLARK

**CREATING SOLID POWER PIVOT DATA MODELS**

Self-service business intelligence is gaining popularity among business analysts today. It greatly relieves the problems created by traditional data warehouse implementations. Using tools such as Microsoft's Power Pivot, Power Query and Power View alters the process significantly. In this session, Dan will take you through the process of creating a solid data model in Power Pivot. You will learn how to import data from various sources, combine these in a scalable data model, use DAX to create measures, and incorporate time-based analysis.

### HOWARD DEINER

**GETTING PAST THE 50 + 70 = 120 CALCULATOR IN CUCUMBER: 12 THINGS TO WORK ON**

With the best intentions, people have flocked to behaviour-driven development by way of Cucumber over the past few years, and that's a great thing! But often, BDD can fall by the wayside due to the pressure to deliver more and more functionality, sprint after sprint. In this session, Howard will explore 12 of the most important issues, such as imperative versus declarative style, and how to keep Gherkin-driven Selenium WebDriver tests working dependably through the use of advanced Expected-Condition techniques.

### NUNO GODHINO

**ARCHITECTURE BEST PRACTICES ON WINDOWS AZURE**

When new technologies and paradigms appear, it's essential to learn them quickly and well. But this can be difficult, since some things are only learned with experience. That's why best practices are so important. In this session, Nuno will look at some architecture best practices that will help us make our solutions better across several levels, including performance, cost, integration, security and so on. By doing this, you'll gain the knowledge needed to quickly start using the technology and paradigms that can help improve your business.

## DAY 3 AGENDA CONTINUED

### 14.00

#### PETER O'HANLON

**BUILDING NATURAL USER INTERFACES WITH REALSENSE DEVICES**

The Intel RealSense SDK is the powerhouse behind the perceptual computing cameras powering the next generation of Ultrabook devices. In this session, Peter will take a look at how RealSense devices can provide unique ways to interact with applications, explaining the advantages, and offering tips and tricks to building compelling applications using RealSense devices. You will learn how to easily create applications that use gesture and facial recognition, emotion detection, as well as speech recognition and speech synthesis.

#### ANDREY ADAMOVICH

**GROOVY DEVOPS IN THE CLOUD**

In this session, Andrey will focus on a set of tools to automate the provisioning of (cloud) servers using Groovy libraries and Gradle plug-ins. He will explore how to leverage those to create an infrastructure for building, configuring and testing the provisioning of boxes in the cloud – elegant and groovy. This session will help those Java/Groovy developers interested in reusing their existing skills for infrastructure provisioning and learning more about problems encountered during system operations.

#### SHAI REZNIK

**18 TIPS FOR THE ANGULAR ARCHITECT**

So your company is planning to build a large-scale web application, and has chosen to do it in Angular.js. That raises a lot of questions: Where do I start from? What tools should I use? And, basically, how do I avoid making mistakes and do the job efficiently? Shai has worked with more than 20 companies, helping them with their struggles migrating to Angular and avoiding crucial mistakes in the process. In this talk, Shai will present useful time-saving architecture tips that will help you prepare for scalability, write cleaner code, and even make your life happier.

#### IDO FLATOW

**DEBUGGING THE WEB WITH FIDDLER**

Every web developer needs to see what goes on "in the wire", whether it is a jQuery call from JavaScript, a WCF service call from a client app, or a simple GET request for a web page. With Fiddler, the most famous HTTP sniffer, this is simple enough to do. But Fiddler is more than just a sniffer. With Fiddler you can intercept, alter and record messages, and even write your own message visualiser. In this session, we will learn how to use Fiddler from bottom to top to debug, test and improve web applications.

#### MICHAEL HABERMAN

**FROM XAML/C# TO HTML/JS**

In recent years, the mobile evolution caused many developers to find themselves migrating from desktop applications to web applications. In this session, Michael will explore how to make the transition from XAML and C# to HTML5 and JavaScript. He will review how to port the MVVM design pattern to the web environment, and go on to tackle important architecture concepts, such as dependency injection and modularity.

### 16.00

#### KEVLIN HENNEY

**GIVING CODE A GOOD NAME**

Code is basically made up of three things: names, spacing and punctuation. With these three tools, a programmer needs to communicate intent, and not simply instruct. But if we look at most approaches to naming, they are based on the idea that names are merely labels, so that discussion of identifier naming becomes little more than a discussion of good labelling. A good name is more than a label; a good name should change the way the reader thinks. Good naming is part of good design. In this session, Kevlin will look at why and what it takes to get a good name.

#### JOE NATOLI

**THE BIG LIE: WHY FORM DOESN'T (AND SHOULDN'T) FOLLOW FUNCTION**

The prescriptive interpretation of this axiom has guided the work of engineers, programmers, developers – and even designers – for a very long time. The result of this has been sites, software and systems that exhibit poor usability, frustrating user experiences and a marked failure to deliver expected business results. In this session, Joe will show you why pure function is rarely the single or most important component of success. He will explain how every force at play in any project is what really evolves form (and dictates function).

#### DINO ESPOSITO

**MAKING AN EXISTING WEB SITE A MOBILE-FRIENDLY WEB SITE**

Do you feel frustrated every time you run across a web site that doesn't adjust to the viewport of your current phone browser? In some cases, for a better experience, you have to know that a mobile version of the site exists somewhere with a different URL. There's no reason to further delay plans to make your primary web site display nicely on small-screen devices, including smartphones. In this session, Dino will lead the discussion and explore pros, cons and technologies that could make each option viable.

#### GIL FINK

**THE CHARACTERISTICS OF A SUCCESSFUL SPA**

Single-page applications (SPAs) are web applications that are built using a single page, which acts as a shell to all the other web pages, with a rich JavaScript front-end. As opposed to traditional web applications, most of the SPA development is done on the front-end. The server, which once acted as a rendering engine, provides only a service layer to the SPA. In this session, Gil will explain the characteristics and building blocks that form the foundation of any successful SPA.

#### NEAL FORD

**BUILD YOUR OWN TECHNOLOGY RADAR**

ThoughtWorks' Technical Advisory Board creates a "technology radar" twice a year: a working document that helps the company make decisions about what technologies are interesting. This is a useful exercise both for you and your company. In this session, Neal will describe the radar visualisation, how to create litmus tests for technologies, and the process of building a radar. Attendees will leave with tools that enhance your filtering mechanisms for new technology and help you (and your organisation) develop a cogent strategy to make good choices.

**ALSO EACH DAY...**

**8.30** COFFEE & REGISTRATION | **11.00** COFFEE BREAK | **13.00** LUNCH BREAK | **15.30** COFFEE BREAK

### SASHA GOLDSHTEIN

**DAWN OF A NEW ERA: AN OPEN-SOURCE .NET**

The .NET framework is now open source, with the CLR to follow along and a cross-platform reference implementation for Linux and OS X to show up during the year. In this session, Sasha will talk about the future of .NET in this new era, what it means for the core stack on the server and desktop, and how it's going to affect our .NET applications. We will also see how to build, test and run our own version of the .NET framework and CLR, and how to make changes to components previously treated as a black box.

### DEJAN SARKA

**VISUALISING GRAPHICAL AND TEMPORAL DATA WITH POWER MAP**

Power Map is a new 3D visualisation add-in for Excel, used for mapping, exploring and interacting with geographical and temporal data. Power Map exists as free preview add-in for Excel 2013 and in the Power BI suite in Office 365. In this session, Dejan will explain how to use Power Map to plot geographic and temporal data visually, analyse that data in 3D, and create cinematic tours to share with others.

### JULES MAY

**PROBLEM SPACE ANALYSIS**

How do you design a large system? The architecture of any system is crucial to its success – get this wrong, and the project may never recover. And yet, we are expected to deliver designs that can last five, 10, sometimes 30 years into an unknowable future. Problem space analysis is a technique that informs and documents system designs by anticipating and defining the variabilities of a long-lived, evolving system. In this session, Jules will explain the principles of the method, give an outline of the benefits, and demonstrate its power with some illustrative examples.

### TUSHAR SHARMER

**DOES YOUR DESIGN SMELL?**

In this session, Tushar will propose a unique approach to developing high-quality software design. Borrowing a phrase from the healthcare domain, "a good doctor is one who knows the medicines but a great doctor is one who knows the disease", the proposed approach is grounded on the philosophy that "a good designer is one who knows about the design principles but a great designer is one who understands the problems (or smells) with the design, their cause, and how they can be addressed by applying proven and sound design principles".

### ANTHONY SNEED

**SOUP TO NUTS: DEVELOPING REAL-WORLD BUSINESS APPS WITH ENTITY FRAMEWORK AND ASP. NET WEB API**

One-day workshop continues from the morning session.

*For a full description of the workshop, please see Page 21*

**ONE-DAY WORKSHOP**

**WORKSHOP REF: MC02**

### ANDY CLYMER & RICHARD BLEWETT

**SOLID ASYNC PROGRAMMING IN .NET**

Two-day workshop continues from the morning session.

*For a full description of the workshop, please see Page 20*

**TWO-DAY WORKSHOP**

**WORKSHOP REF: MC03**

### KLAUS ASCHENBRENNER

**JOINS IN SQL SERVER - AS EASY AS ABC?**

Have you ever looked at an execution plan that performs a join between two tables? And have you ever wondered what a "Left Anti Semi Join" actually is? Joining two tables using SQL Server is far from easy! In this session, Klaus will take a deep dive into how join processing happens in SQL Server. Initially, he will lay out the foundation of logical join processing, then dig deeper into physical join processing in the execution plan. After attending this session, you will be well prepared to understand the various join techniques used by SQL Server.

### GARY SHORT

**TROLL HUNTING ON THE INTERNET**

With so many people on social media these days, almost inevitably not a day goes by without some tragedy befalling someone. As if that wasn't horrible enough, these poor souls and their families can then become victims of the perverse behaviour of the "trolls. In this session, Gary will examine this problem from a data scientist's point of view, showing how to use computational linguistics to ensure that such posts never reach people's streams, and network theory to trace and expose the trolls so that they no longer have the shield of anonymity to hide behind.

### SEB ROSE

**MONAD AT THE HYPERBOLE (AND OTHER AWESOME STORIES)**

In this session, Seb will help you to be a better software developer. As software developers, we have to deliver something useful to our customers. We have to produce it in a manner that acknowledges their requirements and context. And usually, we need to be able to work as part of a team. Are my customers more likely to be satisfied if I'm awesome? In what circumstances would monads (or any other implementation level detail) be a critical part of a successful solution? Seb will analyse real-world examples of projects that succeeded and failed.

### IDO FLATOW

**MIGRATING APPLICATIONS TO MICROSOFT AZURE: LESSONS LEARNED FROM THE FIELD**

How much time will it take us to move to Azure? Can we just "Lift & Shift" our servers? Will my load-balancer work in Azure? Should I use SQL Databases or an SQL Server VM? These are just some of the questions customers ask when they consider migrating their applications to Azure. If you're evaluating Azure, come to this session, where Ido will explain what to do, what not to do, what to avoid and what to embrace when moving your apps to Azure. These are not general best practices; these are lessons learned from the field.

## POST-CONFERENCE WORKSHOPS

**DevWeek concludes with our final day of post-conference workshops.**

**As on previous days, all workshops run for a full day, from 09.30 to 17.30, with short breaks in the morning and afternoon, and a lunch break at 13.00.**

09.30

NEAL FORD

### CONTINUOUS DELIVERY

Getting software released to users can be painful, risky and time-consuming. Here, Neal sets out the principles and practices that enable rapid, incremental delivery of high-quality, valuable new functionality to users. By automating the build, deployment and testing process, and improving collaboration between developers, testers and operations, delivery teams release changes in a matter of hours or even minutes.

Neal will look at the differences between related topics such as continuous integration, continuous deployment and continuous delivery, and explore the new technical artefact that continuous delivery introduces: the deployment pipeline. He'll discuss the various stages, how triggering works, and how to pragmatically determine what "production ready" means, before covering the role of testing and the testing quadrant, including the audience and engineering practices around different types of tests. This is followed by version control usage and offering alternatives to feature branching, such as toggle and branch by abstraction. Neal will then go on to cover operation, DevOps and programmatic control of infrastructure, using tools such as Puppet and Chef.

**WORKSHOP REF: DW11**

JOE NATOLI

### GENERATING MEANINGFUL REQUIREMENTS

Ask any group of people what they want or need and you'll find no shortage of opinions or answers. Clients and stakeholders will always have a voluminous laundry list of features and functions, all of which they will insist are equally important. Your clients, employers, project stakeholders and users all share something very important in common: they're all human beings. And we human beings all have a fundamental flaw: we often make very confident – but equally false – predictions about our future behaviour.

So the requirements that will actually be most useful and most valuable – the ones that will increase user adoption or sales; the ones that will make or save money – are almost never surfaced in traditional requirements sessions.

In this workshop, Joe will show you how to change that, along with how to tell the difference between what people say they need and what they *actually* need. Finally, he'll show you how to uncover the things they don't know they need (but absolutely do).

**WORKSHOP REF: DW12**

JULES MAY

### PROBLEM SPACE ANALYSIS

How do you design a large system? We know Waterfall doesn't work very well; yet we also know that Agile scales poorly. Various proposals have been made (BDUF, domain-driven design, prototyping) but none really solves the problem.

The key to managing a large system is managing *change*. No specification ever survives its own implementation: as a system takes shape, everyone – developers, architects, stakeholders – change their minds. In any non-trivial project, goalposts are constantly in motion. A robust architecture is one that anticipates those changes, and a good design is one that accommodates them cheaply and efficiently.

Problem space analysis is a technique that simply and clearly anticipates, documents and defines the changes that can affect a project. It informs the architectural design so that it can accommodate those changes, and it delivers a change-tolerant ubiquitous language to unify and coordinate the development effort.

Jules introduces the principles of problem space analysis and how they translate into architectures and working systems, even while the goalposts are moving. The technique will be actualised using a real-life design problem.

**WORKSHOP REF: DW13**

TUSHA SHARMA

### ACHIEVING DESIGN AGILITY BY REFACTORING DESIGN SMELLS

Software design plays an important role in adapting changing requirements for software development. Software products that follow Agile methodologies are no exception. Agile methodologies welcome rapid requirement changes and yet promote timely delivery. Often, software teams follow these practices (to some extent) but ignore the software design and its quality due to negligence or ignorance. "Design agility" suggests that the software design has to be Agile if the team intends to follow Agile practices in a true sense. In the absence of agility of design, it's difficult to achieve the benefits of being Agile.

Tushar emphasises the importance of design agility by exposing design smells in software systems. He explains design smells and connects them to their impact on software design, design agility, bug-proneness and delivery schedule of the software.

This workshop offers an understanding of the importance of software design quality and design agility. You'll also learn the vocabulary of smells, using a comprehensive classification, and practical refactoring strategies to repay technical debt.

**WORKSHOP REF: DW14**

> "Great lectures, very informative. Fantastic venue"
>
> JUNIOR WEB DEVELOPER 2014 DELEGATE

**IDO FLATOW**

**THE ESSENTIAL TOOLBOX FOR TROUBLESHOOTING ASP.NET WEB APPLICATIONS**

Is you web application working slower than anticipated? Have you rewritten your application code, but still wonder if there is some ASP.NET or IIS trick you can use to boost things up?

Testing and profiling a web application is trickier than with desktop or mobile apps. It requires testing both front-end and back-end, and the network in-between the two. And since web applications are a mix of HTML, CSS, JavaScript and .NET code, you often need to use several tools to accomplish this task.

In this workshop, Ido will go over the process of testing and profiling web applications, and demonstrate how to use various tools of the trade – both for front and back ends. Ido will then go into detail on how to speed up your web applications, by providing important tips and tricks you can apply to the different parts of the application: ASP.NET (Web Forms, MVC and Web API), general .NET best practices, IIS server, networking tips, HTML/CSS and JavaScript.

**WORKSHOP**
**REF: DW15**

**SASHA GOLDSHTEIN**

**MAKING THE MOST OF C++11/14**

In this workshop, Sasha will look at the most important C++ language features that improve system performance and developer productivity, and see how to apply them to existing code.

The C++11 standard is already behind us, and C++14 is just around the corner. With a huge variety of language features such as lambdas, rvalue references, auto and decltype, and variadic templates, it's easy to get lost in C++. In fact, it often seems like a completely new and foreign language. We will make the most of Visual C++ 2013 and give a special focus to converting and refactoring code to use modern C++ idioms. Specifically, we will look at how to best use STL algorithms with lambda functions, when to use each kind of smart pointer class, how to convert macros and non-generic code to templates, and a variety of best practices concerning concurrency in C++ applications.

This will be a particularly relevant workshop for C++ developers who watched the C++ 11/14 train passing by and weren't able to apply all the best practices of modern C++ to their applications just yet.

**WORKSHOP**
**REF: DW16**

**DEJAN SARKA**

**BI WITH MICROSOFT TOOLS: FROM ENTERPRISE TO A PERSONAL LEVEL**

Microsoft has really been investing a lot in business intelligence (BI) over the last 10 years. The result is a huge number of analytical tools and services. When building a BI solution, many companies make basic mistakes and choose inappropriate tools for the problem they are trying to solve.

In this workshop, Dejan will help you put the building blocks into the right context. You will learn about data warehousing with SQL Server, reporting with SQL Server Reporting Services (SSRS), Power View and Power Map, on-line analytical processing (OLAP) with SQL Server Analysis Services (SSAS), Multidimensional, Tabular, Power Pivot for Excel, and Power Pivot for SharePoint Server, data mining with SSAS, Excel, R and Azure Machine Learning (ML), and about the extract-transform-load process with SQL Server Integration Services (SSIS) and Power Query.

**WORKSHOP**
**REF: DW17**

**GIL FINK**

**BUILDING SCALABLE JAVASCRIPT APPS**

Building and maintaining large and scalable JavaScript web apps isn't easy. So how do you build your front-end-oriented applications without being driven to madness? Using and combining proven JavaScript patterns will do the trick.

In this workshop, Gil will discuss the patterns behind some of the largest JavaScript apps, such as Gmail and Twitter, and we'll explore how to apply them in your own apps. We'll start from object patterns and discuss how to write more object-oriented-like code in JavaScript. Then, we will focus on module patterns and asynchronous module definitions (AMD). We will also discuss patterns such as promises, timers and mediator. At the end of the day, we will combine the patterns and see how to use them to build your next scalable JavaScript web app.

**WORKSHOP**
**REF: DW18**

**ALLEN HOLUB**

**THE SWIFT PROGRAMMING LANGUAGE**

Swift is Apple's new programming language. It represents a significant improvement over both Objective C and C++; incorporating many contemporary language features (such as duck typing and closures) without abandoning object orientation. It promises to become the dominant language of the Apple platform.

Swift is, unfortunately, a mixed bag. It supports many important language features, but it also omits features that are essential for long-term maintainability (privacy, for example).

In this workshop, geared to programmers who already know an OO language (C++, Java, C#, etc.), Allen will present all the interesting parts of the Swift language. We'll gloss over the basic stuff (declarations, flow control, etc.) and focus on those parts of the language that will be new to you (lambdas, subscripts, the inheritance model, extensions and chaining, etc.), with considerable emphasis on places where the language can get you into trouble.

**WORKSHOP**
**REF: DW19**

**SEB ROSE**

**APPLIED BDD WITH CUCUMBER/JAVA AND SPECFLOW/C#**

It's all very well reading books, but nothing beats actually getting practical experience. In this workshop, working in your choice of Java, C# or Ruby, Seb will drive out the implementation of a simple utility by specifying its behaviour in Cucumber. The tyrannical Product Owner will regularly change his mind, so we'll need to keep our code well factored and easy to modify.

This session is designed for developers and testers. By the end of the day, you'll be comfortable working with Cucumber in your chosen development environment. You'll have seen, first hand, how to use Cucumber to drive out valuable features for your customers and how that can help keep your stakeholders engaged in the software development process. It will also be clear how BDD interacts with TDD.

Bring a laptop with your chosen development environment installed, and try to pre-install your chosen Cucumber variant before you come (instructions available). But don't worry, we can install on the day if necessary.

If this subject interests you, but you'd prefer a gentler intro, don't miss Seb's earlier workshop on Monday: "BDD by example".

**WORKSHOP**
**REF: DW20**

# DevWeek's main conference days now offer more choice than ever before. Alongside the breakout sessions, you can choose from a special two-day workshop that runs across both Wednesday and Thursday, and one-day workshops to choose from on each day.

**TWO-DAY WORKSHOP:**

**WED & THURS**

ANDREW CLYMER

RICHARD BLEWETT

## SOLID ASYNC PROGRAMMING IN .NET

**In this special two-day workshop, Andrew and Richard will take you through the core skills required to be successful developing async and multithreaded code, both in the .NET and web worlds. Not only do we cover the core APIs but also how they are used effectively, tested and debugged.**

**Tasks**
When the Parallel Framework Extensions (PFx) were first announced, it looked as though they were going to target a narrow set of requirements around parallelising processor intensive code. Over time, the scope of the library has grown significantly, such that it will become the main model for building asynchronous code. The pivotal type enabling this transition is the Task class. This is a functionally very rich type, allowing the creation of both short- and long-lived asynchronous work, Tasks can have dependencies on one another and support cancellation. In this, the first of the PFx modules we look specifically at how this class gives us a unified framework for building multithreaded code.

**Thread safety**
Asynchronous programming requires careful attention to detail, since most objects are not designed with multithreaded access in mind. This module introduces the importance of Interlocked and Monitor-based synchronisation.

**Concurrent data structures**
Ever since its inception, .NET has had support for a number of synchronisation primitives (such as Interlocked, Monitor and Mutex). However, on their own, these primitives do not provide support for more complex synchronisation situations, and so people have had to use them as building blocks to build things such as efficient semaphores. PFx finally brings to the library a set of richer primitives, such as lazy initialisation, a lightweight semaphore and a countdown event. But more than this, it also introduces a set of high-performance concurrent data structures that allow you to use them without you having to provide your own synchronisation logic around them. We also look at the new Immutable Collections package, now available on NuGet, which provides another way to model data for async applications.

**Parallel**
The initial goal of PFx was to simplify the parallelisation of processor intensive tasks – and this remains a key feature. This part of its functionality is focused on the Parallel class and its For and ForEach members. In this module, we look at the simplified model but also highlight that parallelising algorithms is never as simple as it might first seem – we show you some of the pitfalls that you should be aware of when trying to parallelise functionality using the Parallel class.

**async/await**
C# 5 builds on the Task API, introducing async and await keywords, which bring asynchronous execution as a first-class concept in the C# language. These new keywords create a very elegant model for all sorts of async work and this module explains not only how to use them but also how they work under the covers.

**Server-side async**
The nature of server-side applications often means they are asynchronous by their very design, servicing many clients at the same time on different threads. But as you dig deeper, you often find these threads performing long-running blocking operations – particularly in terms of IO resulting in consuming more threads (an expensive resource) than necessary. This module focuses on a range of server-side technologies and demonstrates how to perform maximum concurrency for least number of threads.

**TPL Dataflow**
TPL Dataflow is a downloadable addition to the TPL (Task Parallel Library) that ships with the .NET framework. TPL Dataflow provides an alternative approach to define concurrency. Instead of just simply throwing threads at a synchronously structured program and having to deal with all the thread safe and race conditions that introduces; we have the concept of many autonomous objects, each with its own thread of execution. These autonomous objects co-operate with other such objects through asynchronous message passing. In this module, we will see how TPL Dataflow can greatly reduce the complexity normally associated with asynchronous programming.

**Rx**
Reactive Framework is a new library that uses the .NET 4.0 IObservable interface and LINQ to create a compelling new programming model that allows you to build "event"-based code with declarative LINQ statements. This module introduces the Reactive Framework and shows how it can greatly simplify your code.

> "A wonderful way to rethink what it means to be a great developer"
>
> SOFTWARE DEVELOPER 2014 DELEGATE

**ALSO EACH DAY...**

**8.30** COFFEE & REGISTRATION | **11.00** COFFEE BREAK | **13.00** LUNCH BREAK | **15.30** COFFEE BREAK

**ONE-DAY WORKSHOPS:**

## WEDNESDAY

SAHIL MALIK

### LEARN ANGULARJS: THE ROAD TO POWERFUL, MAINTAINABLE APPLICATIONS

**JavaScript, by its nature, makes it difficult to write maintainable code. HTML, by its nature, is loosely structured. AngularJS fixes both of those. It is a structural framework for dynamic web apps, allowing you to extend HTML's syntax, enabling you to write powerful, maintainable applications succinctly.**

In this workshop, Sahil will build on your existing knowledge of JavaScript and teach you the ins and outs of AngularJS. There are plenty of examples, which will walk you through a basic introduction, models, controllers and views in Angular, templates and databinding, services and dependency injection, directives, routing and single-page applications.

**An introduction to AngularJS**
AngularJS is a structural framework for dynamic web apps. It lets you use HTML as your template language and lets you extend HTML's syntax to express your application's components clearly and succinctly. This module gets you started with AngularJS showing the basic syntax and some quick starts to get us running.

**Models, controllers and views**
Controllers in AngularJS are a fundamental building block of Angular. AngularJS encourages better code architecture by encouraging you to use MVC-based patterns. There is rich support for controllers and views, and this module will familiarise you with those.

**Templates and databinding**
When working with complex forms, or even reactive user interfaces, it really helps to leverage concepts such as databinding. Databinding in JavaScript is especially valuable but can be difficult to implement, unless you use something like Angular.

**Services in Angular, dependency injection**
Services in Angular are substitutable objects that are wired together using dependency injection. Fancy! What on Earth does that even mean? Well for one, services are a classic way of bundling together reusable code across controllers in your application, and they can be lazily instantiated or singletons. There are services that Angular provides, such as $http, or ones you can create. And then there is something called interceptors. This module gets in in the deep of AngularJS services.

**Directives**
Directives in AngularJS are markers on DOM elements that tell the AngularJS HTML compiler to attach some specified behaviour to that particular DOM element or its children. There are many directives, such as ngBind, ngModel, ngClass, and this module shows you the most important directives.

Routing and single-page applications (SPAs)
AngularJS routes enable you to create different URLs for different content in your application. Having different URLs for different content enables the user to bookmark URLs to specific content. Single-page applications allow you to write HTML and JavaScript code that more or less performs and behaves like a thick client application. It's pretty neat – trust me, you'll be using this.

> **Very informative, covering a broad range of technology and a good opportunity to engage with industry experts**

SOFTWARE ARCHITECT, 2014 DELEGATE

## THURSDAY

ANTHONY SNEED

### SOUP TO NUTS: DEVELOPING REAL-WORLD BUSINESS APPS WITH ENTITY FRAMEWORK AND ASP.NET WEB API

**Performance. Scalability. Maintainability. Testability. Security. Today's application developers need to build systems that are designed to achieve these goals from the outset. In this in-depth workshop, Anthony will take you beyond the basics, to learn what it takes to build RESTful services that are robust, scalable and loosely coupled, using dependency injection with repository and unit of work design patterns.**

But there's more to building loosely coupled systems than applying a set of design patterns. Anthony will show how you can harness the power of code generation by customising T4 templates for reverse engineering Code First classes from an existing database, in order to produce entities with persistence concerns that are completely stripped away. You'll also learn ninja techniques for handling cyclical references with code-based configuration and using efficient binary formatters, all without polluting your entities with mapping, serialisation or validation attributes.

This workshop will focus primarily on developing real-world business apps using the Entity framework and ASP.NET Web API.

**DevWeek's speakers are acknowledged experts in their field. Recognised internationally, the 2015 speaker faculty comprises professional consultants, trainers, industry veterans, thought-leaders and published authors. Find out more about them here.**

## SPEAKER PROFILES

### ANDREY ADAMOVICH

**Andrey** is a software craftsman with years of experience. His true love is the JVM ecosystem, and applying it to his company's DevOps initiatives. He is one of the authors of the *Groovy 2 Cookbook*, and a frequent speaker at conferences.

### KLAUS ASCHENBRENNER

**Klaus** provides independent SQL Server Consulting Services across Europe and the US. He has worked with SQL Server 2005/2008/2012/2014 from its very beginning, and has also written the book *Pro SQL Server 2008*.

### AUSTIN BINGHAM

**Austin** is a founder of Sixty North, a Norway-based software consulting, training and application development company. Developer of industry-leading oil reservoir modelling software in C++ and Python, he is an experienced presenter and teacher.

### RICHARD BLEWETT

**Richard** has worked on distributed systems, including as middle-tier architect on the UK national police systems. He focuses on technologies that enable developers to build large-scale systems on the Microsoft platform, such as WCF, BizTalk, Workflow and Azure.

### PEARL CHEN

**Pearl**'s cross-disciplinary approach ranges from Android to Arduino, HTML to LEDs. Her work has taken her from Facebook campaigns for Google Chrome to projects that turn payphones into gumball machines or dynamically create origami from SMS messages.

### AMY CHENG

**Amy** is a web developer at the Brooklyn Museum of Art in New York City, working to increase dialogue between the museum and its visitors through technology, and has been a mentor for the non-profit Girls Who Code. She is interested in using code to create art.

### DAN CLARK

**Dan** is a senior business intelligence (BI)/ programming consultant specialising in Microsoft technologies. A former physics teachers, he has written several books and numerous articles on .NET programming and BI development, and is a regular conference speaker.

### ANDREW CLYMER

**Andy** is a co-founder of Rock Solid Knowledge, creating Kiosk-based solutions on Windows Embedded with .NET. He cut his teeth programming on a host of platforms at various start-ups, and now consults and teaches for a diverse range of clients.

### ED COURTENAY

**Ed** is an experienced software developer and technical evangelist who has been programming professionally for more than 25 years. He currently works for a major manufacturer and retailer in the UK, leading the team responsible for its ecommerce web site.

### HOWARD DEINER

**Howard** is a software consultant and educator who specialises in Agile process and practices. With a career spanning more than 30 years, he's been a developer, analyst, team lead, architect and project manager, and is a long-standing member of the ACM and IEEE.

### DINO ESPOSITO

**Dino** is a trainer, speaker, consultant and author. CTO of Crionet, a company providing software and mobile services to professional sports, Dino is also technical evangelist for software developer JetBrains, focusing on Android and Kotlin.

### GIL FINK

**Gil** is a web development expert, ASP.NET/ IIS Microsoft MVP and the founder of sparXys. He consults for various enterprises and companies, where he helps to develop web and RIA-based solutions, and conducts lectures and workshops.

### IDO FLATOW

**Ido** is a senior architect and trainer at SELA Group, a Microsoft ASP.NET/IIS MVP, and an expert on Microsoft Azure and web technologies such as WCF, ASP.NET and IIS. He has co-authored a number of books and official Microsoft courses.

### NEAL FORD

**Neal** is director, software architect and meme wrangler at ThoughtWorks, a global IT consultancy focusing on end-to-end software development and delivery. He's the author of applications, articles, and books on a variety of subjects and technologies.

### SHAY FRIEDMAN

**Shay** is a Visual C#/ IronRuby MVP and the author of *IronRuby Unleashed*. With more than 10 years' experience in the software industry, he is the co-founder of CodeValue, a company that creates products for developers, consults and conducts courses.

### NUNO GODINHO

**Nuno** is Director of Cloud Services, Europe at Aditi Technologies, and has more than 16 years' experience in IT. His specialities include enterprise architecture and solution architecture, cloud computing, development and training.

### SASHA GOLDSHTEIN

**Sasha** is the CTO of Sela Group, a Microsoft C# MVP and Azure MRS, a Pluralsight author, and an international consultant and trainer. The author of two books, Sasha is a prolific blogger and author of numerous training courses.

### MICHAEL HABERMAN

**Michael** (MCT, MCPD) is a senior consultant and lecturer specialising in rich client technologies such as WPF, Windows Phone, XNA and HTML/JS. He has helped to develop complex infrastructures using Prism, MVVM and Angular.

### DROR HELPER

**Dror** is a senior consultant at software company CodeValue, with a decade of experience ranging from Intel and SAP to small start-ups. He evangelises Agile methodologies and test-driven design in his work, at conferences and as a consultant.

### SANDER HOOGENDOORN

**Sander** is the author of the best-selling book *This Is Agile*. An independent mentor, trainer, programmer, architect, speaker and writer, Sander is a catalyst in the innovation of software development at many international clients.

### ALLAN KELLY

**Allan** has held just about every job in software, before joining Software Strategy to help teams adopt and deepen Agile practices. He has written books including *Xanpan – Team-centric Agile Software Development*, and is a regular speaker and journal contributor.

### MICHAEL KENNEDY

**Michael** is an author, instructor and technical curriculum director at DevelopMentor, and lead developer for its online training platform, LearningLine. He has been building commercial applications with .NET since its initial public beta in 2001.

### TOBIAS KOMISCHKE

**Tobias** is Senior Director of User Experience at Infragistics, Inc., and has worked in user experience for more than 10 years. He specialises in Human Factors Engineering, which is rooted in his academic background in cognitive psychology.

### IQBAL KHAN

**Iqbal** is the President and Technology Evangelist of software developer Alachisoft, maker of NCache, the industry's leading open-source distributed cache for .NET. NCache is also available for Microsoft Azure.

### PHIL LEGGETTER

**Phil** is a Developer Evangelist at Caplin Systems, working on the BladeRunnerJS open source project. He writes frequently and specialises in JavaScript development and real-time web technologies.

### SAHIL MALIK

**Sahil**, the founder and principal of Winsmarts.com, has been a Microsoft MVP and INETA speaker for 11 years. Author of books and articles about Microsoft technologies, iOS and JavaScript, Sahil helps make the most difficult topics fun.

## KEYNOTE SPEAKERS

KEVLIN HENNEY

**Kevlin** is an independent consultant and trainer based in the UK. His development interests are in patterns, programming, practice and process. He is co-author of two volumes in the Pattern-Oriented Software Architecture series, editor of the book *97 Things Every Programmer Should Know*, and a columnist for various magazines and web sites.

ALLEN HOLUB

**Allen** is an internationally recognised consultant, trainer, speaker and author. He specialises in lean/Agile processes and culture, Agile-focused architecture and cloud-based web-application development. He has written a dozen books, hundreds of magazine articles, and currently blogs on Agile for Dr Dobb's Journal.

### CHRISTOS MATSKAS

**Christos** is a software engineer with more than 10 years' experience mainly focusing on the .NET stack. He has worked with big names including MarkIT, Strathclyde University, Amor/Lockheed Martin, Ignis Asset Management and Barclays.

### JULES MAY

**Jules** is a software architect with a particular interest in languages (both for programming and discourse), presently active in web and mobile convergence. He has been writing, teaching and speaking for 25 years, and is the originator of "Problem Space Analysis".

### JAMES MONTEMAGNO

**James** is a Developer Evangelist at Xamarin. He has been a .NET developer for more than a decade, working in industries including games development, printer software and web services, with several published apps on iOS, Android and Windows.

### JOE NATOLI

**Joe** has been preaching and practising the gospel of user and customer experience to Fortune 100, 500 and government organisations for more than 25 years. As founder of Give Good UX, he offers coaching, training and product audit programmes.

### PETER O'HANLON

**Peter**'s fascination with new technologies has seen him writing articles for CodeProject, blogging and contributing to open-source projects. He was recently made an Intel Software Innovator for his work with RealSense technology.

### JOHN K. PAUL

**John** is the VP of engineering at Penton Media and former lead technical architect of Condé Nast's platform engineering team. He also organises the NYC HTML5 meetup group, and contributes to a number of open-source projects.

### TONI PETRINA

**Toni** is a Microsoft MVP for C#, developer, speaker, blogger and technology enthusiast. With years of professional experience working on range of technologies, his recent focus has been on Windows Phone and Windows 8 as a platform.

### SHAI REZNIK

**Shai** is an AngularJS consultant working with enterprise companies, helping with migration and building large-scale projects. He recently founded HiRez.io, an online training web site teaching front-end architecture with humour.

### SEB ROSE

**Seb** focuses on helping teams adopt and refine their Agile practices. The founding trainer with Kickstart Academy, he has more than 30 years' industry experience (including IBM Rational and Amazon), and is a popular speaker at international meetings.

### OREN RUBIN

**Oren** has more than 16 years' experience with IBM, Cadence, Wix and others, and is the founder of Testim.io. He regularly speaks about new technologies in web development and test automation, and teaches at the Techion, Israel institute of Technology.

### DEJAN SARKA

**Dejan**, MCT and SQL Server MVP, is an independent consultant, trainer and developer focusing on database and business intelligence applications. He specialises in topics like data modeling, data mining and data quality, and has written 13 books.

### TUSHAR SHARMA

**Tushar** is a technical expert at the Siemens Research and Technology Center in Bangalore, India. His research into software design, design smells and refactoring has resulted in several patents, research papers and tools.

### PAVEL SKRIBTSOV

**Pavel** is a graduate of the Moscow Institute of Physics and Technologies (MIPT), with a PhD in Neurocomputer application for the representation of static and dynamic 3D data. He is also the founder, CEO and ideological leader of Pawlin Technologies Ltd.

### GARY SHORT

**Gary** is a freelance data science practitioner and trainer. He has a deep understanding of the full Hadoop and HDInsight environment, as well as an interest in Social Network Analysis, (UCINet and Pajek) and computational linguistics (NLTK).

### ROBERT SMALLSHIRE

**Robert** is a founding director of Sixty North, a software product and consulting business in Norway. He has worked in senior architecture and technical management roles, providing tools for dealing with the masses of information flowing from today's energy sector.

### MARK SMITH

**Mark** runs the curriculum team at Xamarin University, building and managing the growing course catalogue used to train Xamarin developers all over the world. He is a Microsoft MVP, Wintellect author and Xamarin Consulting partner.

### ANTHONY SNEED

**Tony** is a course author, instructor and consultant for Wintellect, specialising in robust, scalable and maintainable applications using Entity framework, WCF, Windows Identity Foundation and ASP.NET Web API. He is the author of two popular open-source frameworks.

### ADAM TORNHILL

**Adam** combines degrees in engineering and psychology for a different perspective on software. An architect and programmer, he writes open-source software in a variety of languages, and is the author of *Your Code as a Crime Scene*.

### RALPH DE WARGNY

**Ralph** has spent 10 years focusing on the software development market at Intel, helping companies and institutions to maximise application performance and move code from serial to parallel. He has spoken at TechEd, TechDays, OOP and other conferences across Europe.

### MIKE WOOD

**Mike** is a Technical Evangelist for Red Gate Software, on the Cerebrata Team. He describes himself as a "problem-solving, outdoorsy, user group founding, dog-loving, blog writing, solution-creating, event planning, married, technology-speaking, father-of-one".

### EOIN WOODS

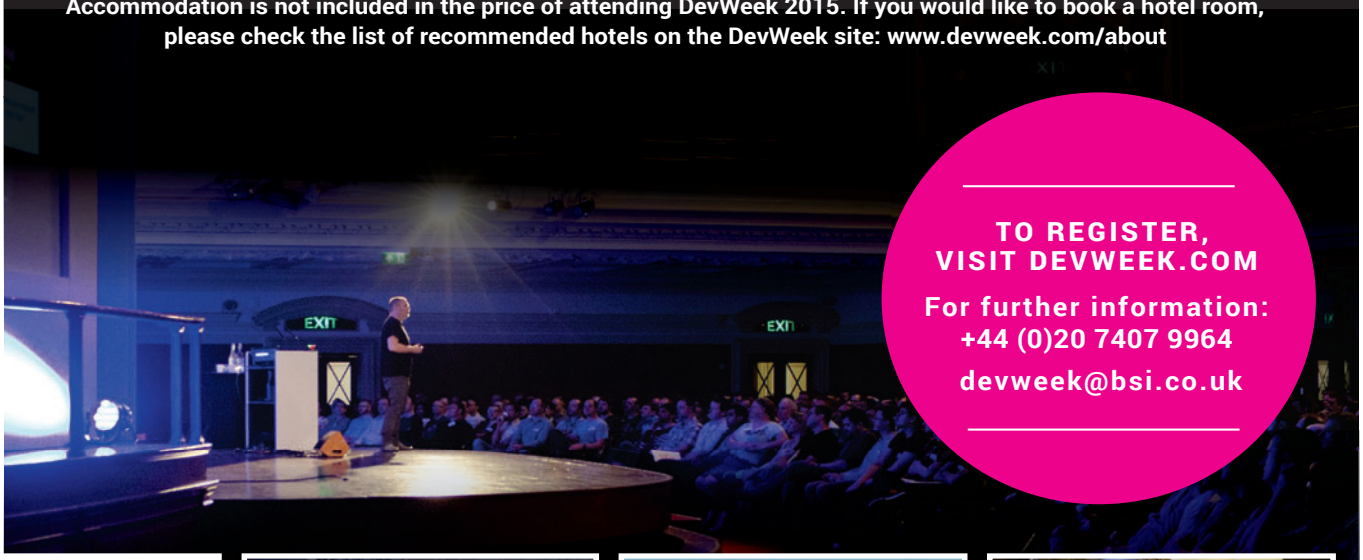
**Eoin** is a lead architect in the Operations Technology group at UBS. Prior to UBS, he spent 20 years in software engineering at Bull, Sybase, InterTrust and BGI. His main interests are software architecture, distributed systems, computer security and data management.

## DEVWEEK 2015 PRICING STRUCTURE

BOOK NOW

| All prices include refreshments, buffet lunch and session notes, but exclude travel and accommodation | BOOK BY FRIDAY 19 DECEMBER 2014 SAVE UP TO £300 | BOOK BY FRIDAY 30 JANUARY 2015 SAVE UP TO £200 | BOOK BY FRIDAY 6 MARCH 2015 SAVE UP TO £100 | BOOK AFTER FRIDAY 6 MARCH 2015 |
|---|---|---|---|---|
| UNIVERSAL PASS ALL 5 DAYS | £1,495 +VAT | £1,595 +VAT | £1,695 +VAT | £1,795 +VAT |
| 4-DAY PASS MAIN CONFERENCE + PRE/POST WORKSHOP | £1,195 +VAT | £1,295 +VAT | £1,395 +VAT | £1,495 +VAT |
| 3-DAY PASS MAIN CONFERENCE ONLY | £895 +VAT | £995 +VAT | £1,095 +VAT | £1,195 +VAT |
| 2-DAY WORKSHOP PASS MAIN CONFERENCE 2-DAY WORKSHOP | £645 +VAT | 745 +VAT | £845 +VAT | £945 +VAT |
| 1-DAY WORKSHOP PASS PRE/POST MAIN CONFERENCE WORKSHOP | £345 +VAT | £395 +VAT | £445 +VAT | £495 +VAT |

**Accommodation is not included in the price of attending DevWeek 2015. If you would like to book a hotel room, please check the list of recommended hotels on the DevWeek site: www.devweek.com/about**




**TO REGISTER, VISIT DEVWEEK.COM**

**For further information:**
**+44 (0)20 7407 9964**
**devweek@bsi.co.uk**


**TERMS AND CONDITIONS:** Please note that the online submission of a completed registration form constitutes a firm booking, subject to the following terms and conditions. Any cancellations received after Friday 16 January 2015 will incur a 30% administration fee. Cancellations must be made in writing at least 60 days before the conference, or the full fee will be charged. We are happy to accept substitutions if they are submitted in writing before the conference begins. The organisers reserve the right to make changes to the programme and speakers without notice, if this is unavoidable. If delegates are unable to attend for any reason that is beyond the control of the organisers, such as transport problems, personal illness, bereavement, inclement weather, terrorism or Act of God, it will not be possible to make any refunds of conference or workshop fees.