THE MICROSOFT JOURNAL FOR DEVELOPERS

OCTOBER 2015 VOL 30 NO 10

msdn magazine

THE MICROSOFT JOURNAL FOR DEVELOPERS

OCTOBER 2015 VOL 30 NO 10

msdn magazine

Understanding Cloud Development.....12

COLUMNS

Microsoft

msdn magazine

OCTOBER 2015 VOLUME 30 NUMBER 10

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.

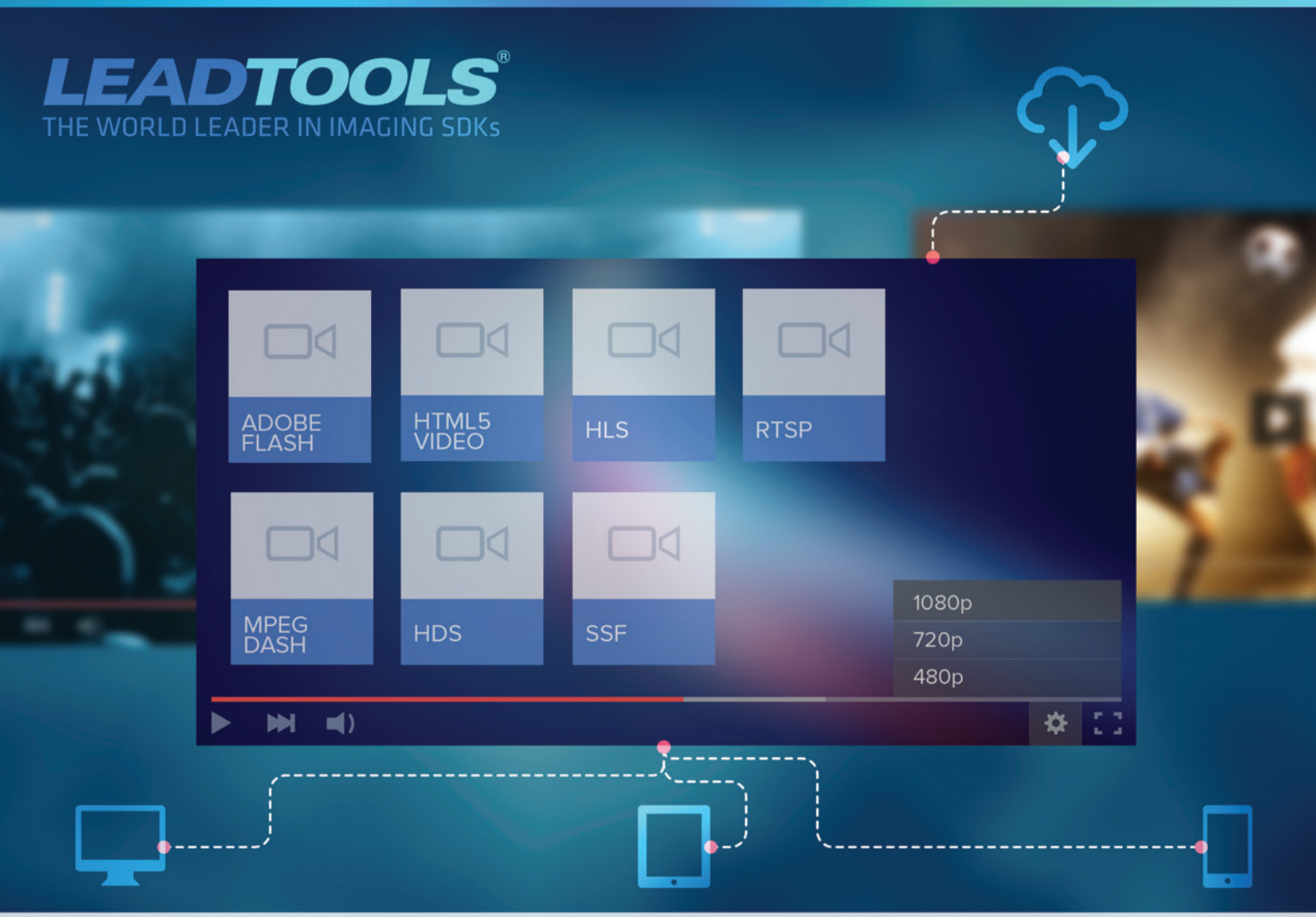
Microsoft

BPA WORLDWIDE

1906 100 2006 AMERICAN BUSINESS MEDIA

WPA Western Publications Association

LEADTOOLS®
THE WORLD LEADER IN IMAGING SDKs



QUICKLY BUILD PROFESSIONAL AUDIO/VIDEO MULTIMEDIA STREAMING APPLICATIONS FOR PLAYBACK ON ANY DEVICE

Media Streaming Server SDK

- *High-level Media Streaming Server framework with minimal coding required*
- *Place any audio/video media file on the server and stream to any client*
- *Automatically convert videos on the fly from a single base format to minimize recompression*
- *Broadcast audio/video content compressed with LEADTOOLS' industry-leading compression technology*
- *Includes .NET (C# & VB) and C DLL libraries for 32 and 64-bit development*

DOWNLOAD OUR 60 DAY EVALUATION
WWW.LEADTOOLS.COM

LEAD TECHNOLOGIES INCORPORATED

SALES@LEADTOOLS.COM
800.637.1840

MICHAEL DESMOND

Chain of Disaster

The discipline of computer security has more than a little in common with airline flight safety. Both are fraught with high stakes and increasing complexity over time. When something goes wrong, remediation is often built on painstaking forensics and paid for with funds that become available only *after* a high-profile failure. Another common thread: Catastrophic failure often springs from mundane causes.

Take the infamous data breach at Target in 2013. Hackers entered the network the old-fashioned way—they stole credentials from an HVAC contractor with login rights to the Target network, and from there gained access to payment systems. The installation of malware was actually detected and flagged by the FireEye security software Target had deployed months earlier, yet when security staff in Bangalore forwarded the alerts to Minneapolis, the security team there declined to take action. Over the months that followed, some 40 million debit and credit cards, along with gigabytes of customer data, were exfiltrated from the Target systems, leading to more than $100 million in losses for the retail giant.

A similar "chain of disaster" pattern is evident in many airliner accidents. Air France 447 in 2009 crashed into the Atlantic Ocean after encountering thunderstorms at night near the equator while enroute from Rio de Janeiro to Paris. It was later learned that ice had likely blocked the aircraft's pitot tubes—small openings on the side of the hull for measuring airspeed and pressure. This produced incorrect and conflicting data that disabled the plane's autopilot and apparently disoriented the first officer flying the craft through the storm. Possibly convinced that his plane was flying dangerously fast, he commanded a constant nose up attitude that, in fact, sharply reduced airspeed and produced a high-altitude stall. The aircraft ultimately fell belly first into the sea, killing all on board.

In both cases, systems that performed as designed in the face of negative events were misinterpreted by the people managing them. Both the security team at Target and the pilot on Air France 447 struggled to comprehend the data presented to them and took actions that made a bad situation worse. Part of the blame rests with the human operators, but part lies also with the systems themselves.

In both cases, systems that performed as designed in the face of negative events were misinterpreted by the people managing them.

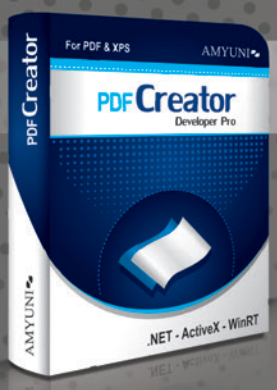
For the crew of Air France 447, confusing audible warnings played a role. By design, the stall warning of the Airbus 340 goes silent if measured airspeed falls below a threshold where the data is considered invalid. So when the first officer lowered the nose of the jet to gain much-needed airspeed, it actually caused the stall warning to reengage, while pulling back to raise the nose (and resume the deep stall) silenced the alarm. Faced with conflicting data inputs and confusing feedback from the stall horn, the pilots very likely didn't know what instruments to trust.

If there is one overriding parallel between air accident investigations and software security incidents, it's that each presents an invaluable opportunity to better understand the complex interaction of human factors, environmental stresses, and system behavior and automation.

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

Switch to Amyuni PDF

AMYUNI



Create & Edit PDF's in .Net - ActiveX - WinRT - Universal Apps

NEW v5.5

- 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 Universal Apps DLLs enable publishing C#, C++CX or Javascript apps to windows Store
- Updated 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

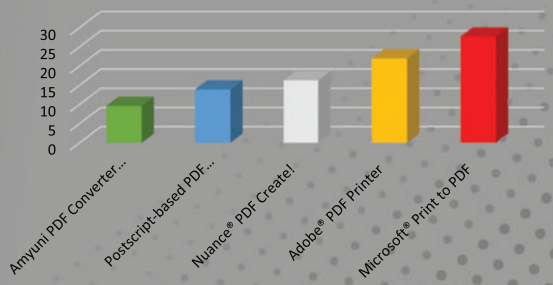
Other Developer Components from Amyuni

- WebkitPDF: Direct conversion of HTML files into PDF and XAML without the use of a web browser or a printer driver
- PDF2HTML5: Conversion of PDF to HTML5 including dynamic forms
- Postscript to PDF Library: For document workflow applications that require processing of Postscript documents
- OCR Module: Free add-on to PDF Creator uses the Tesseract engine for character recognition
- Javascript engine: Integrate a full Javascript interpreter into your applications to process PDF files or for any other need

High Performance PDF Printer for Desktops and Servers

- Print to PDF in a fraction of the time needed with other tools. WHQL tested for all Windows platforms. Version 5.5 updated for Windows 10 support

Benchmark Testing - Amyuni vs Others
Seconds required to convert a document to PDF



CERTIFIED FOR Windows Server 2008

Windows Server 2012 Certified

Windows 7

Windows 8

AMYUNI Technologies

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

All development tools available at
www.amyuni.com

The Yoga of Rookie Success

As a young engineer, I struggled at first to fit into my team. After all, my colleagues had been there for years. How could I ever compare? The more I advanced, the more my teammates seemed to stay ahead of me. Being intelligent wasn't enough. Maybe I'd made a terrible mistake in becoming an engineer, I thought.

In cracking this puzzle, I discovered that the best way to move from novice to expert was to do the thankless tasks—for example, helping with operational improvements. I had avoided these activities because they didn't get my heart racing like R&D work. Why would any "real" engineer want to maintain a build validation system or create a testing framework? But it was precisely by serving my team that I could transcend my own narrow realm of expertise. I did tasks nobody else wanted: migrating infrastructure systems, resolving product support backlogs, talking to external vendors, investigating performance regressions. Along the way, I elevated my whole team, gained broad expertise, and built strong personal relationships with the veterans whose experience had intimidated me.

By doing these tasks well, I also gained a reputation as an efficient, trustworthy and reliable engineer. The juicy work I'd wanted, but hadn't felt ready for in the past, now came automatically. I had matured and proven that I wasn't obsessed with just myself.

I discovered that the best way to move from novice to expert was to do the thankless tasks—for example, helping with operational improvements.

Other positive changes emerged. When I focused on others' interests rather than my own, I blasted through roadblocks and gained expertise rapidly. When my motivation was to help, I felt comfortable approaching others with "stupid" questions and even socializing with them. Being selfless was simply more efficient!

I began operating at a new level. No longer trapped in an obsessive race for fast promotions and more money, I made it my goal to help others, and soon those same rewards started to come easily. Today, if another colleague serves the team better, I see it as an opportunity to contribute to her success and to learn from her.

A Beginner's Mind

For a long time, I felt insecure about how much I didn't know. The more knowledge I consumed, the more expansive the universe of things I *didn't* know became. I wondered if a different industry, less subject to drastic shifts, would suit me better.

For a long time, I felt insecure about how much I didn't know. The more knowledge I consumed, the more expansive the universe of things I *didn't* know became.

I struggled with the challenge of mastering a parade of new frameworks and languages every year. My solution? I came to terms with my lack of knowledge and sense of inadequacy.

For example, when asked to lead my team's iOS efforts, I told my manager that I was excited about the opportunity but intimidated by my limited knowledge of the platform. I mastered iOS a little bit at a time, and treated situations that exposed my iOS ignorance as opportunities to enhance my knowledge so I could be more valuable to the team. I was open about what I didn't know, and would often say, "I don't know, but I'll find out." (And I always did.) I found that my team appreciated my vulnerability and follow-through.

I used to operate under the illusion that there's a benefit to being a know-it-all. There *isn't*. It's OK to not know 90 percent of a framework, extension, language, area, product or tool. I know what's relevant to me now, and I'm OK with having a "beginner's mind" for months or years—it keeps me curious and sharp! There's always more to know in our field. Instead of feeling driven by my innate desire to master it all, I trust my intuition and explore only what is driven by a sense of peace of mind. ■

KRISHNAN RANGACHARI *has worked in diverse roles at large software companies and many Silicon Valley start-ups. He writes, speaks and advises on his passion: rigorous career transformation for impatient high-achievers. Reach him at k@radicalshifts.com.*

Of drivers who own family cars,

86%

would rather be ***driving one of these.***

Move into the Fast Lane with MobileTogether®

In a little over two days, with one developer, we created a full-featured survey application and asked Android™, iOS®, and Windows® mobile users about their dream cars. 86% wanted to go faster.

ALTOVA®
MobileTogether®

Ditch the Minivan of Mobile Development

It's no longer practical to wait weeks or months for a cutting-edge mobile business solution. Your team needs to create, test, and deploy to all devices at the speed of business. With MobileTogether, we've removed the roadblocks.

- Drag and drop UI design process
- Powerful visual & functional programming paradigm
- Native apps for all major mobile platforms
- Connectivity to SQL databases, XML, HTML, and more
- Deploy full-featured business app solutions in record time
- Pricing so affordable you might just have money left to start saving for that sports car

www.altova.com/MobileTogether

KENNY KERR

Coroutines in Visual C++ 2015

I first learned about coroutines in C++ back in 2012 and wrote about the ideas in a series of articles here in *MSDN Magazine*. I explored a lightweight form of cooperative multitasking that emulated coroutines by playing clever tricks with switch statements. I then discussed some efforts to improve the efficiency and composability of asynchronous systems with proposed extensions to promises and futures. Finally, I covered some challenges that exist even with a futuristic vision of futures, as well as a proposal for something called resumable functions. I would encourage you to read these if you're interested in some of the challenges and history related to elegant concurrency in C++:

- "Lightweight Cooperative Multitasking with C++" (msdn.com/magazine/jj553509)
- "The Pursuit of Efficient and Composable Asynchronous Systems" (msdn.com/magazine/jj618294)
- "Back to the Future with Resumable Functions" (msdn.com/magazine/jj658968)

The C++ Standard Library offers futures and promises in an attempt to support asynchronous operations, but they've been much maligned due to their naïve design.

Much of that writing was theoretical because I didn't have a compiler that implemented any of those ideas and had to emulate them in various ways. And then Visual Studio 2015 shipped earlier this year. This edition of Visual C++ includes an experimental compiler option called /await that unlocks an implementation of coroutines directly supported by the compiler. No more hacks, macros or other magic. This is the real thing, be it experimental and as yet unsanctioned by the C++ committee. And it's not just syntactic sugar in the compiler front end, like what you find with the C# yield keyword and async methods. The C++ implementation includes a deep engineering investment in the compiler back end that offers an incredibly scalable implementation. Indeed, it goes well beyond what you might find if the compiler front end simply provided a more convenient syntax for working with promises and futures or even the Concurrency Runtime task class. So let's revisit the topic and see what this looks like today. A lot has changed since 2012, so I'll begin with a brief recap to illustrate where we've come from and where we are before looking at some more specific examples and practical uses.

I concluded the aforementioned series with a compelling example for resumable functions, so I'll start there. Imagine a pair of resources for reading from a file and writing to a network connection:

```
struct File
{
  unsigned Read(void * buffer, unsigned size);
};

struct Network
{
  void Write(void const * buffer, unsigned size);
};
```

You can use your imagination to fill in the rest, but this is fairly representative of what traditional synchronous I/O might look like. File's Read method will attempt to read data from the current file position into the buffer up to a maximum size and will return the actual number of bytes copied. If the return value is less than the requested size, it typically means that the end of the file has been reached. The Network class models a typical connection-oriented protocol such as TCP or a Windows named pipe. The Write method copies a specific number of bytes to the networking stack. A typical synchronous copy operation is very easy to imagine, but I'll help you out with **Figure 1** so that you have a frame of reference.

As long as the Read method returns some value greater than zero, the resulting bytes are copied from the intermediate buffer to the network using the Write method. This is the kind of code that any reasonable programmer would have no trouble understanding, regardless of their background. Naturally, Windows provides services that can offload this kind of operation entirely into the kernel to avoid all of the transitions, but those services are limited

Figure 1 **Synchronous Copy Operation**

```
File file = // Open file
Network network = // Open connection

uint8_t buffer[4096];

while (unsigned const actual = file.Read(buffer, sizeof(buffer)))
{
  network.Write(buffer, actual);
}
```

Figure 2 **Copy Operation with Futures**

```
File file = // Open file
Network network = // Open connection

uint8_t buffer[4096];

future<void> operation = do_while([&]
{
  return file.ReadAsync(buffer, sizeof(buffer))
    .then([&](task<unsigned> const & read)
    {
      return network.WriteAsync(buffer, read.get());
    })
    .then([&](task<unsigned> const & write)
    {
      return write.get() == sizeof(buffer);
    });
});

operation.get();
```

to specific scenarios and this is representative of the kinds of blocking operations apps are often tied up with.

The C++ Standard Library offers futures and promises in an attempt to support asynchronous operations, but they've been much maligned due to their naïve design. I discussed those problems back in 2012. Even overlooking those issues, rewriting the file-to-network copy example in **Figure 1** is non-trivial. The most direct translation of the synchronous (and simple) while loop requires a carefully handcrafted iteration algorithm that can walk a chain of futures:

```
template <typename F>
future<void> do_while(F body)
{
  shared_ptr<promise<void>> done = make_shared<promise<void>>();
  iteration(body, done);
  return done->get_future();
}
```

The algorithm really comes to life in the iteration function:

```
template <typename F>
void iteration(F body, shared_ptr<promise<void>> const & done)
{
  body().then([=](future<bool> const & previous)
  {
    if (previous.get()) { iteration(body, done); }
    else { done->set_value(); }
  });
}
```

The lambda must capture the shared promise by value, because this really is iterative rather than recursive. But this is problematic as it means a pair of interlocked operations for each iteration. Moreover, futures don't yet have a "then" method to chain continuations, though you could simulate this today with the Concurrency Runtime task class. Still, assuming such futuristic algorithms and continuations exist, I could rewrite the synchronous copy operation from **Figure 1** in an asynchronous manner.

Figure 3 **Copy Operation within Resumable Function**

```
future<void> Copy()
{
  File file = // Open file
  Network network = // Open connection

  uint8_t buffer[4096];

  while (unsigned copied = await file.ReadAsync(buffer, sizeof(buffer)))
  {
    await network.WriteAsync(buffer, copied);
  }
}
```

I would first have to add async overloads to the File and Network classes. Perhaps something like this:

```
struct File
{
  unsigned Read(void * buffer, unsigned const size);
  future<unsigned> ReadAsync(void * buffer, unsigned const size);
};

struct Network
{
  void Write(void const * buffer, unsigned const size);
  future<unsigned> WriteAsync(void const * buffer, unsigned const size)
};
```

The WriteAsync method's future must echo the number of bytes copied, as this is all that any continuation might have in order to decide whether to terminate the iteration. Another option might be for the File class to provide an EndOfFile method. In any case, given these new primitives, the copy operation can be expressed in a manner that's understandable if you've imbibed sufficient amounts of caffeine. **Figure 2** illustrates this approach.

The do_while algorithm facilitates the chaining of continuations as long as the "body" of the loop returns true. So ReadAsync is called, whose result is used by WriteAsync, whose result is tested as the loop condition. This isn't rocket science, but I have no desire to write code like that. It's contrived and quickly becomes too complex to reason about. Enter resumable functions.

Adding the /await compiler option enables the compiler's support for resumable functions, an implementation of coroutines for C++. They're called resumable functions rather than simply coroutines because they're meant to behave as much like traditional C++ functions as possible. Indeed, unlike what I discussed back in 2012, a consumer of some function shouldn't have to know whether it is, in fact, implemented as a coroutine at all.

As of this writing, the /await compiler option also necessitates the /Zi option rather than the default /ZI option in order to disable the debugger's edit-and-continue feature. You must also disable SDL checks with the /sdl- option and avoid the /RTC options as the compiler's runtime-checks aren't compatible with coroutines. All of these limitations are temporary and due to the experimental nature of the implementation, and I expect them to be lifted in coming updates to the compiler. But it's all worth it, as you can see in **Figure 3**. This is plainly and unquestionably far simpler to write and easier to comprehend than what was required for the copy operation implemented with futures. In fact, it looks very much like the original synchronous example in **Figure 1**. There's also no need in this case for the WriteAsync future to return a specific value.

The await keyword used in **Figure 3**, as well as the other new keywords provided by the /await compiler option, can appear only within a resumable function, hence the surrounding Copy function that returns a future. I'm using the same ReadAsync and WriteAsync methods from the previous futures example, but it's important to realize that the compiler doesn't know anything about futures. Indeed, they need not be futures at all. So how does it work? Well, it won't work unless certain adapter functions are written to provide the compiler with the necessary bindings. This is analogous to the way the compiler figures out how to wire up a range-based for statement by looking for suitable begin and end functions. In the case of an await expression, rather than looking for begin and end, the compiler looks for suitable functions called await_ready, await_suspend and await_resume. Like begin and end, these new functions may be either member functions or free functions. The ability to write non-member functions is tremendously helpful as you can then write adapters for existing types that provide the necessary semantics, as is the case with the futuristic futures I've explored thus far. **Figure 4** provides a set of adapters that would satisfy the compiler's interpretation of the resumable function in **Figure 3**.

Again, keep in mind that the C++ Standard Library's future class template doesn't yet provide a "then" method to add a continuation, but that's all it would take to make this example work with today's compiler. The await keyword within a resumable function effectively sets up a potential suspension point where execution may leave the function if the operation is not yet complete. If await_ready returns true, then execution isn't suspended and await_resume is called immediately to obtain the result. If, on the other hand, await_ready returns false, await_suspend is called, allowing the operation to register a compiler-provided resume function to be called on eventual completion. As soon as that resume function is called, the coroutines resume at the previous suspension point and execution continues

Figure 4 **Await Adapters for a Hypothetical Future**

```
namespace std
{
  template <typename T>
  bool await_ready(future<T> const & t)
  {
    return t.is_done();
  }

  template <typename T, typename F>
  void await_suspend(future<T> const & t, F resume)
  {
    t.then([=](future<T> const &)
    {
      resume();
    });
  }

  template <typename T>
  T await_resume(future<T> const & t)
  {
    return t.get();
  }
}
```

Figure 5 **Hypothetical await_suspend Overload**

```
template <typename T, typename F>
void await_suspend(future<T> const & t, F resume, same_thread_t const &)
{
  ComPtr<IContextCallback> context;
  check(CoGetObjectContext(__uuidof(context),
    reinterpret_cast<void **>(set(context))));

  t.then([=](future<T> const &)
  {
    ComCallData data = {};
    data.pUserDefined = resume.to_address();

    check(context->ContextCallback([](ComCallData * data)
    {
      F::from_address(data->pUserDefined)();
      return S_OK;
    },
    &data,
    IID_ICallbackWithNoReentrancyToApplicationSTA,
    5,
    nullptr));
  });
}
```

on to the next await expression or the termination of the function.

Keep in mind that resumption occurs on whatever thread called the compiler's resume function. That means it's entirely possible that a resumable function can begin life on one thread and then later resume and continue execution on another thread. This is actually desirable from a performance perspective, as the alternative would mean dispatching the resumption to another thread, which is often costly and unnecessary. On the other hand, there might be cases where that would be desirable and even required should any subsequent code have thread affinity, as is the case with most graphics code. Unfortunately, the await keyword doesn't yet have a way to let the author of an await expression provide such a hint to the compiler. This isn't without precedent. The Concurrency Runtime does have such an option, but, interestingly, the C++ language itself provides a pattern you might follow:

```
int * p = new int(1);
// Or
int * p = new (nothrow) int(1);
```

In the same way, the await expression needs a mechanism to provide a hint to the await_suspend function to affect the thread context on which resumption occurs:

```
await network.WriteAsync(buffer, copied);
// Or
await (same_thread) network.WriteAsync(buffer,
copied);
```

By default, resumption occurs in the most efficient manner possible to the operation. The same_thread constant of some hypothetical std::same_thread_t type would disambiguate between overloads of the await_suspend function. The await_suspend in **Figure 3** would be the default and most efficient option, because it would presumably resume on a worker thread and complete without a further context switch. The same_thread overload illustrated in **Figure 5** could be requested when the consumer requires thread affinity.

This overload retrieves the IContext-Callback interface for the calling thread (or apartment). The continuation then eventually calls the compiler's resume function from this same context. If that happens to be the app's STA, the app could happily continue interacting with other services with thread affinity. The ComPtr class template and check helper function are part of the Modern library, which you can download from github.com/kennykerr/modern, but you can also use whatever you might have at your disposal.

I've covered a lot of ground, some of which continues to be somewhat theoretical, but the Visual C++ compiler already provides all of the heavy lifting to make this possible. It's an exciting time for C++ developers interested in concurrency and I hope you'll join me again next month as I dive deeper into resumable functions with Visual C++. ■

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 @kennykerr.*

Thanks to the following Microsoft technical expert for reviewing this article: *Gor Nishanov*

Microsoft Azure—the Big Picture

Tony Meleg

Every month, this magazine delves into the details of some new or interesting service in Microsoft Azure. There's a never-ending stream of things developers apparently must know about, but at the same time confusion around all the different ways of accomplishing the same thing. It's not easy to put all these pieces together and see the "big picture." With the pace of innovation our industry is currently experiencing, failing to see the big picture presents a real challenge for IT organizations and IT practitioners alike. Well, this article is all about helping you make sense of Azure without drilling into any specific service—the opposite of what we normally do.

Let's face it, developers aren't always very good at making complex things simple. Usually we take some simple problem and implement a very complex solution. We need to understand how things work, especially things we didn't build ourselves. This is partly a trust issue, but it's also so we can understand how to tweak and fine-tune a component or a whole piece of software for our particular needs. It's a control thing, as well.

Microsoft's Big Bet

If my assertions about developers ring a bell, then you're not going to like what I'm going to tell you next. Azure is a collection of application building blocks or "services," each providing a different capability you might need sometime in an application you want to build. Microsoft believes these blocks should be inherently resilient, highly scalable and self-managing. You can provision any service anywhere in the world, pay only for what you consume and, most important, be able to change your consumption anytime, anywhere. Here's the bit you won't like: These services just work. They abstract you away from the complexity; you don't have much control over them; you can't see inside of them. In other words, you have to just trust them. Doesn't sound like something you might be interested in, does it, because you want control, you don't "trust" and you like complexity?

You can roughly segment Azure into three layers—the datacenter infrastructure, infrastructure services and platform services, as shown in **Figure 1**.

You should think of these three layers as stacked on top of each other, with each layer an abstraction of the layer below. So, Infrastructure as a Service (IaaS) is largely an abstraction of the underlying physical servers, storage and networking. Platform as a Service (PaaS) is an abstraction of the application software infrastructure you'd normally install on servers and use when you create applications, such as Web servers, databases, messaging systems and identity infrastructure. It's the service's job not to provide this software for you, but to give you an entire running, resilient, always-on service (and work behind the scenes to monitor and fix any issues transparently without your application skipping a beat).

These services are available not only in the Azure public cloud, but anywhere. This could be your datacenter, a host or an outsourcer. Microsoft recently announced Azure Stack to make this possible—think packaging up the Azure secret sauce and services

This article discusses:

- Platform services
- The Azure datacenter infrastructure
- Infrastructure services

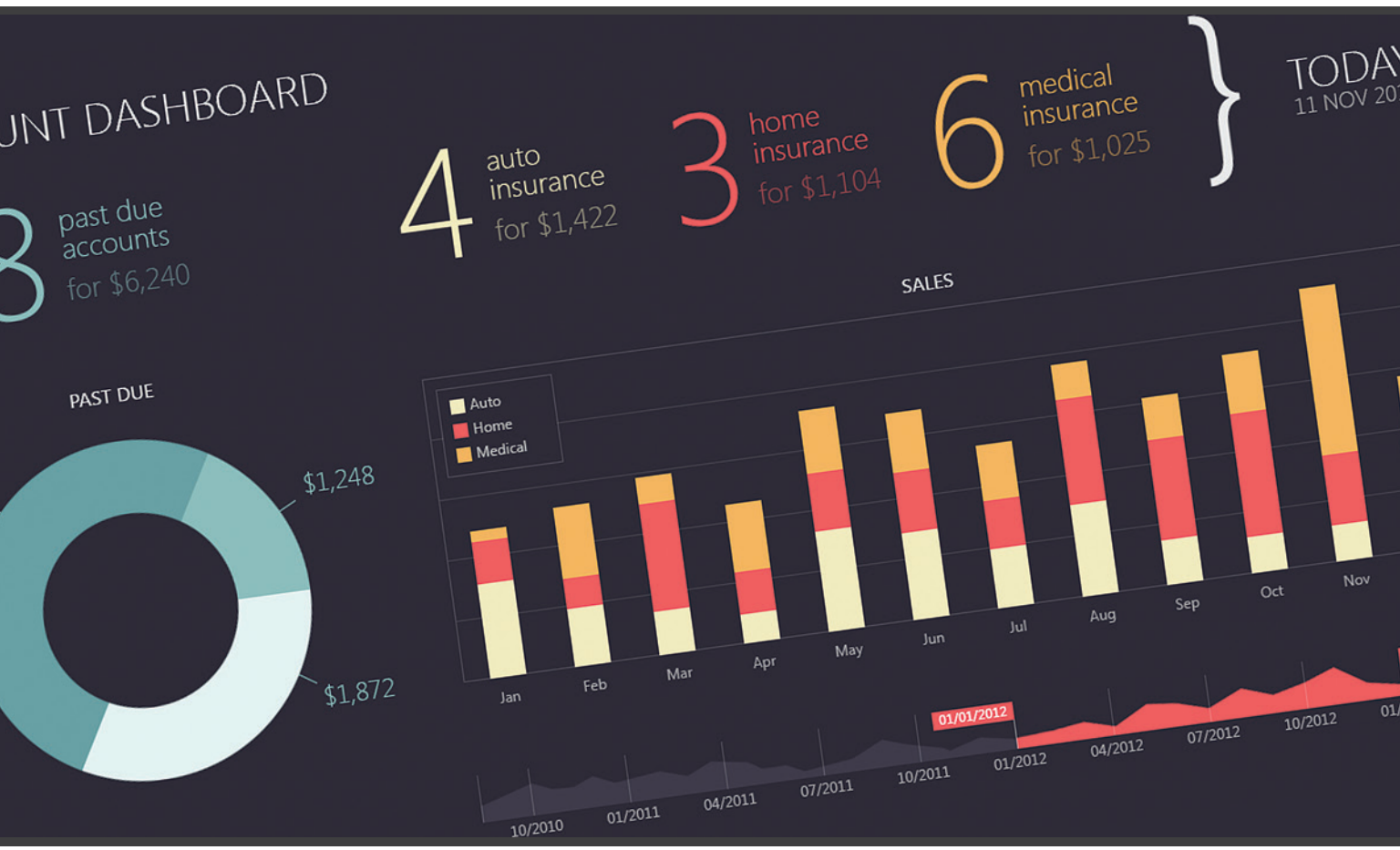
Technologies discussed:

Microsoft Azure

DevExpress®

Dashboards & Analytics

DevExpress Universal ships with everything you'll need to create information-rich decision support systems and to distribute your dashboard solutions royalty-free.



Your Next Great Dashboard Starts Here

Learn how you can create fully customizable Dashboards with our flexible data visualization tools.
devexpress.com/dashboard

and deploying this on your own hardware. After all, it's just software built on top of the core foundations of Windows Server and Hyper-V. In this new world, though, you don't really care because your focus is just on the services you need, how to architect your solution properly and how to program against the services.

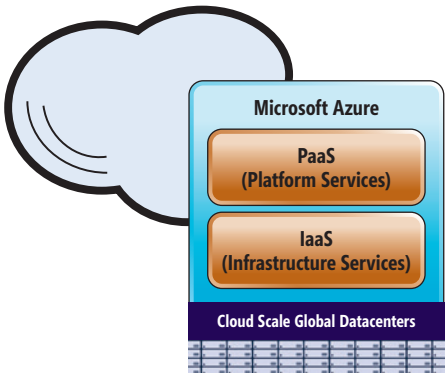


Figure 1 **Microsoft Azure Conceptual View**

The Platform Building Blocks

So let's take a broad look across all of Azure. **Figure 2** lays out all the services in a set of capability groups split across infrastructure services and platform services. You should think of infrastructure services as those capabilities that allow you to create the low-level infrastructure for your existing applications. You need computers that have disks; you need to network them together; you need shared disks; and you need to connect them to other systems and networks in other places and so on.

At the infrastructure services level, the abstractions are more familiar because they represent the underlying physical datacenter, such as a computer in the form of a virtual machine (VM) and virtual disks that you attach to the machine. This allows you to run systems you already have and do things the same way you do today because the only abstraction is the computer, not the application software you might use. It's still your responsibility to install, manage, patch, fix and make resilient any software you run in or across a set of network-connected VMs, just as you do today in your own datacenter.

Platform services, in contrast, involve capabilities that distance you from the lower levels of the computer/storage/network the service is utilizing; that's all abstracted away from you. As a general rule, if you can't actually touch or connect to the underlying VM in a particular service, the service is PaaS. Also, you don't get to choose the software that provides the service, and you don't get to tune or control that software. It's the services job to just work, to fix and patch itself and to monitor all the parts of the service needed across all the Azure datacenters. You get to focus on just using it—which is really what your job is, anyway.

An Aside—the Azure Datacenter Infrastructure

You might be interested, because we're all geeks at heart, in all the underlying complex computing and datacenter infrastructure that powers Azure. You might want to understand the physical hardware specs of the servers, what switches are used, how the racks are built. You might want to know about the complex network topology that provides crazy fast consistent speed across all services and the global network that connects Azure datacenter regions together. Maybe you're curious about all the 24 datacenter regions around the world where Azure exists today (or will soon, as the five new regions in Canada and India come online), as shown in **Figure 3**.

This is your first test as a developer in the new world: You have to stop caring about that, at least in Azure. If you wanted to build

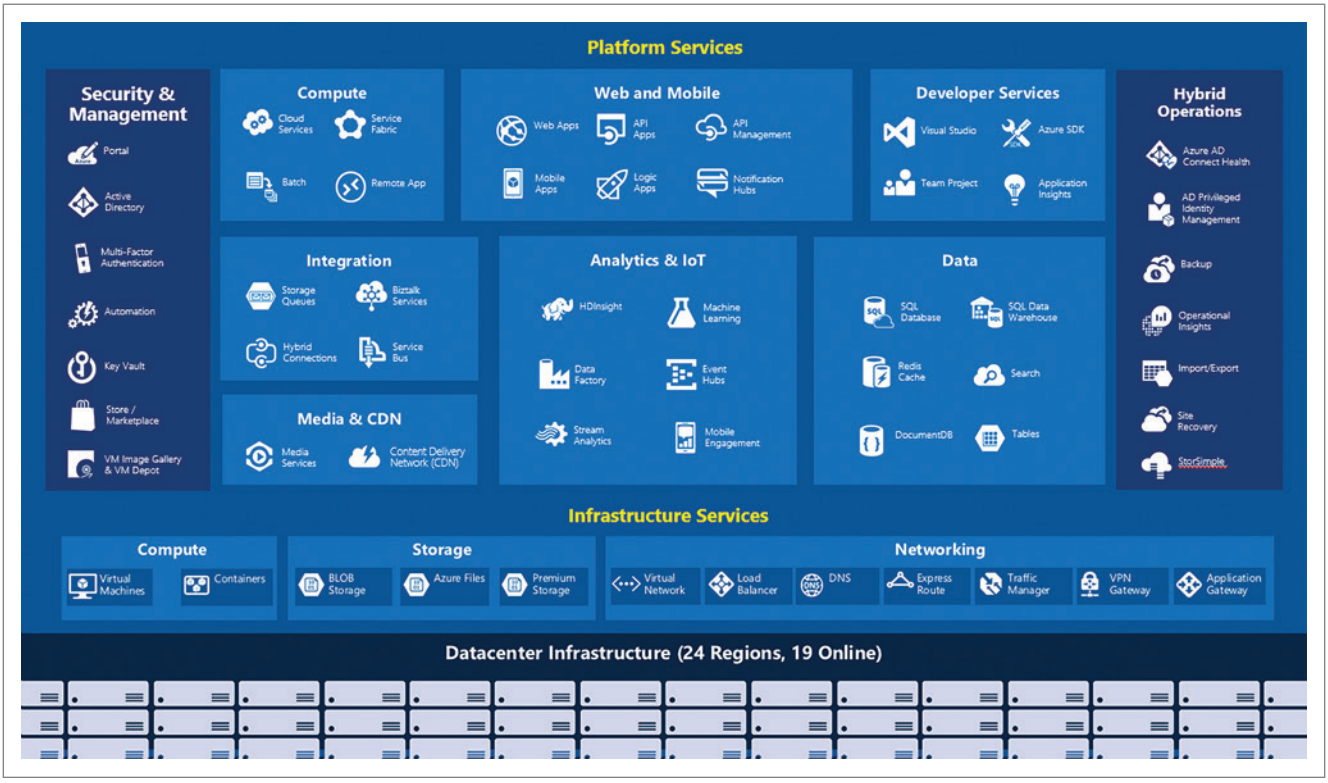
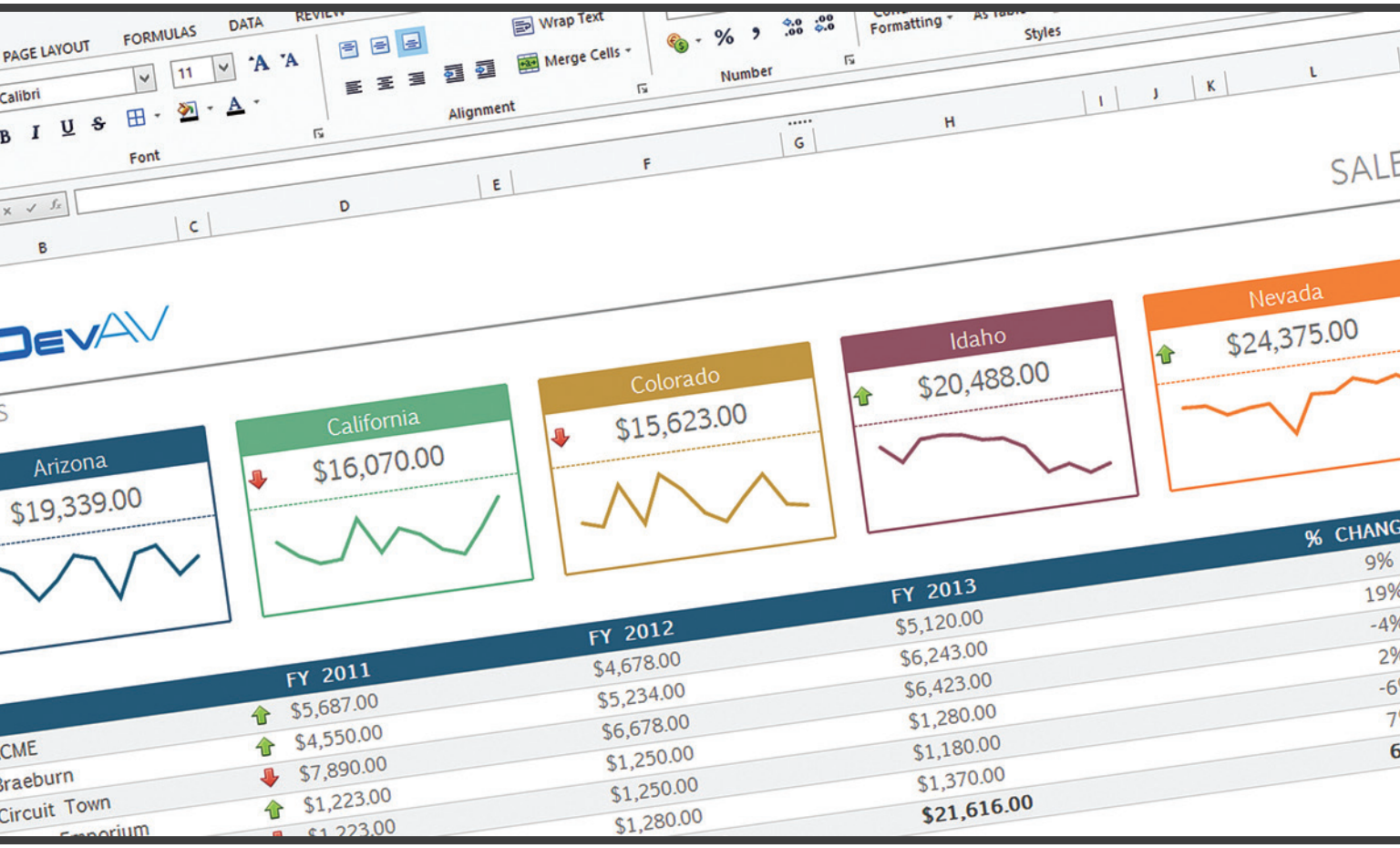


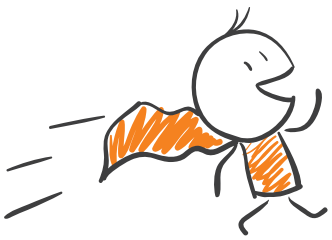
Figure 2 **Microsoft Azure Services**

DevExpress®

Office® Inspired Apps

Get started today and create high-performance, high-impact .NET solutions that fully replicate the look, feel and user-experience of Microsoft Office.®





Your Next Great Business App Starts Here

Explore our complete range of Office-inspired controls for all major .NET platforms.
devexpress.com/office

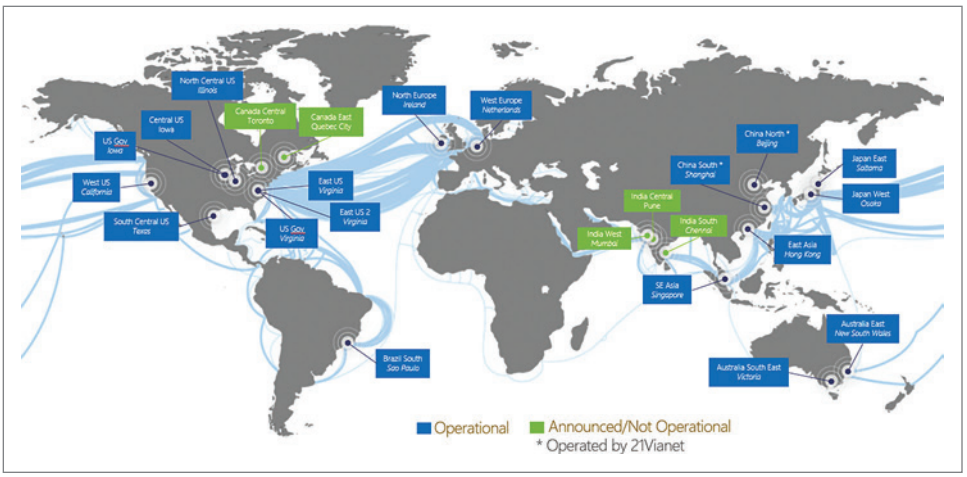


Figure 3 **Global Microsoft Azure Datacenter Footprint**

your own "Azure" in your own datacenter with the Azure Stack, then you (or someone in your organization) would still have to care about all these physical things. One of the reasons you had to care in the past, even about the infrastructure needed for a particular application, was because the cost of getting things wrong was too high. You never really knew the number and size of servers needed; you assumed the worst possible case and added a bit more, just in case. In Azure, those problems don't exist because if you need more of something you just provision more. If you need a bigger machine, you get one; if you get too much, you just change it. In Azure, you just need to work out where your application users are and which are the best datacenters to use to deliver the solution. Usually it's the closest one.

Infrastructure Services—More Control, More Work

When you look across the 60 or so services in **Figure 2**, it's a bewildering list. But most of the apps you've built, most of the time, contain only a few capabilities. Usually there's a Web server and a relational database at the core, plus a bunch of other pieces for identity, reporting and maybe some batch processes. Let's just focus on the Web infrastructure and the database for now.

Straightaway, you have a choice to make: Do you work at the infrastructure or the platform level of abstraction? Do you spin up some VMs, install your Web server and your database and just get going? Remember, this abstraction is primarily about the physical servers, storage/disks and network but not the software you need to run on those servers. Sounds easy, sounds familiar, and—guess what—it is. It's exactly what you do and have done for your entire career. Take a look at **Figure 4**. A VM has virtual disks and these can be shared across VMs or attached to a specific machine. VMs can be put inside virtual networks so they can communicate and networks can be connected to your own datacenter, as well as across Azure regions. This is all done through this software abstraction and it means you end up with a computer that has one or more disks and sits on a network connected to other computers.

Everything inside the set of VMs you need, you get to control and tune and tweak and mess with. In fact, it's even better than your own on-premises world, because of the abstraction. You don't care anymore about the physical machine, the host OS, the hypervisor. You don't have to worry about the resilience of the underlying storage infrastructure for your virtual disks. There's an almost mind-boggling set of choices of VM sizes (different combinations of CPU and RAM, with different disk sizes and speeds and network bandwidths).

Better still, it's all self-service, and everything can be automated so it's incredibly easy to define the infrastructure you need and spin up hundreds of instances with a simple script.

Let's say your app requires a decent level of scale and resilience. It's easy to create a Web farm of VMs and Web servers and create a database cluster. You know (or your IT Pro best friend knows) how to do all this—it's not easy but it's doable. You can do it any time, day or night; you can provision it anywhere in the world; you pay only for what you use and at any time you can change your mind, tear it all down and pay absolutely nothing. How awesome is that? But it sounds too good to be true, so what's the catch?

The catch, at the infrastructure level, is that you still have to do a lot of work to get everything deployed and running. Once it's all running, you have to do even more work to keep it running. There is help, though, especially for the "get it running" part, deriving from the on-premises world where it's even more problematic to deploy a complex set of interrelated VMs, including capabilities in Azure such as Azure Resource Manager and automation through Windows PowerShell, as well as many other solutions from third parties. Once it's all working, you still have to patch all the software you're using inside the VMs, including whatever OS you're running. You

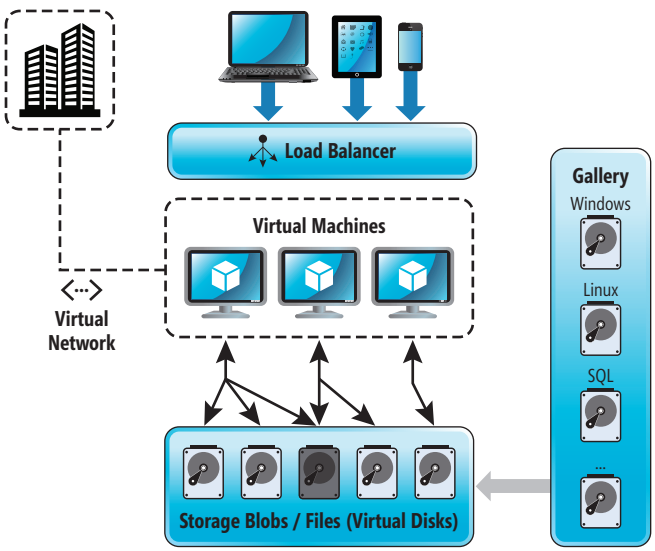


Figure 4 **Virtual Machine Abstraction of Servers, Disks and Networking**

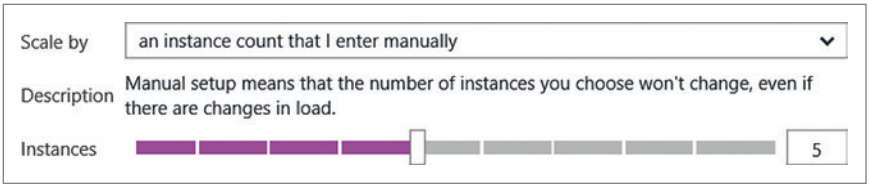


Figure 5 **Changing Web App Instance Count**

have to manage the health of all the application software, which isn't too difficult for the Web server and database, but much more difficult when you scale this out or add in five other capabilities.

You're trading control against work or effort. If you want control, you have to expend more effort building what you need and keeping it all working.

Platform Services—Less Control, Less Work

You might remember the saying "There's no problem in computer science that can't be solved by adding another level of indirection to it." Well, there's another phrase I've seen added to this, along the lines of "… but this will usually create another problem."

It's true, and platform services are a great example. The Azure Services picture in **Figure 2** shows the two platform service abstractions for the two building blocks needed in the example—a Web server and a database. In Azure, these abstractions are Web Apps (in the Web and Mobile section) and SQL Database (in the Data section). You provision these services in Azure in the same way you provision any service, by going to the Azure Management Portal, selecting the service you want and filling in the required details: service name, datacenter location, initial performance/pricing level and so on.

Here's where the less control/less work thing comes into play. Let's take the database, for example. I provision a database in the North Europe datacenter. In less than a minute, I get a fully working database and then it's my job to deploy my schema and data to the database and hook up my Web app. There's no software to install; I get what I get, which is a SQL database. In fact, I don't just get a database, I get three of them, all working together in a highly available cluster, giving me incredible levels of resilience. I can dial the performance of the three databases up and down simply by selecting a different performance level (which changes the price I pay). You can't choose not to have this three-way clustered database; you just get it because the service always wants to make sure it can give you a working database and this is the best way to do that.

As you can see in **Figure 5**, there's a similar model in play when you provision the Web infrastructure for your app using Azure Web Apps. You're able to select the number of instances you want using a simple slider. You dial the instance count up or down and the service does the rest. You can even tell the system to do this for you based on load or schedule. Most important, it's the service's job to always provide the instance count you specified and to monitor and fix instances that are broken. Your job is simply to write your Web app code and hand the bits to the service so it can make sure they're all the places they need to be across your instances.

So, the abstraction for platform services generally provides a service that takes care of the provisioning, resilience and management of the service for you. The "additional problem" that was created with the abstraction is that you lose some control. You can't choose the software, and you can't get "inside the box" and tweak or tune the software being used from the OS all the way up the stack—it's a black box to you. The other control you lose is that you're now locked into the capabilities of that service, as well as the API for the service; that is, how your application code interacts with the service. Maybe you need something specific, for example, in the database, that the service doesn't provide. Maybe it's a specific data type or even an entire capability like full-text search. Maybe you're moving an existing application to the cloud and the capabilities used in your app don't match the capabilities of the service in Azure. What do you do?

The Azure services building blocks picture in **Figure 2** layered all the services into either IaaS or PaaS, but don't think you can't use these "together" and cross that boundary, because you can. There's no reason, for example, that you couldn't use the PaaS Web App service with, say, a VM where you installed Oracle running on Linux or SQL Server running on Windows. Now you have a balance of control and effort. In fact, you can take this even further. Because all of these building blocks are just there waiting to be used, you can, of course, use them from anywhere. All it takes is a simple API or some existing protocol you already use to talk to these services. You can use any of these blocks with your existing systems in your own datacenter. You can use these blocks with other blocks from third parties and even from other cloud providers. This is assuming, of course, an application can tolerate the latency challenges you might introduce, but it's possible.

The Other Services

So why do you need all these other capabilities in Azure? Well, if you look around your own IT shop, across all the applications that are running in production today, you'll find quite a broad collection of application software powering these apps—messaging systems, data analysis capabilities, identity infrastructure, security layers, backup systems, data warehouses, monitoring and management systems, and so on. Collectively, across your entire IT portfolio, you need all this "stuff." I like to think of Azure as a collection of "IT legos." You won't need every type of block all the time as you're building the next great thing, but you do need them all in your toolkit.

How do you decide which blocks to use when you're building a new solution? It's really no different than it's ever been—you have to understand what the building blocks do and match them to the specific needs of your application, and make a call. The perception is that building applications in the cloud is different and more complex. It's neither, but there are some additional things you have to do because of the fundamental concept that you're working in a "shared" infrastructure. This means many services have constraints and these constraints will differ depending on the various functional levels and pricing tiers you're using; you need to know what these are. Remember, you don't want these to be constant; you want to vary what you need based on application load and demand. You have to program defensively for these constraints, as well as service availability and responsiveness. You can use the same tools, the same languages and frameworks, to build solutions, and all services at the lowest level have REST-based

APIs plus framework-specific wrappers to use. This means you can get to the building blocks literally from anything that has a simple Web stack—like a sensor, a reason the cloud is driving the explosion of Internet of Things solutions.

The upside in the cloud is that you can iterate fast (because it's easy to spin up the services); you can try some alternatives; and you can fail fast and cheaply because you've invested very little. Look back at **Figure 2**. Is anything missing? You might see things in the picture you think you'll never need, but I bet you'd have a hard time finding some capability you need or already have that's not represented. There's a lot of guidance out there to help you, broad guidance and detailed implementation guidance service by service.

It's in the DNA of most developers, liking to learn new things. Chances are, you like to play with technology, to try new things. Even if your company is never in a million years going to build anything and put it in a public cloud, it's still the world's best developer playground for you to learn and grow.

Dealing with Change

I want to deal with one final topic before finishing and that's all about dealing with change. I have to admit, I find it hard to keep up with all the innovation that's being delivered in Azure at the crazy pace it's happening, and it's sort of my job to keep up with it. Gone are the days when you had a three- to five-year roadmap of features and capabilities that would be delivered. These days it's six months if you're lucky. You'll find that during the course of building something, new capabilities, even whole new services, appear and it's tempting to evaluate and refactor these into your solution. Capabilities and services can also be removed with a one-year notice period, not the 10-plus year support cycle that's traditionally been in place for server software.

Service versioning is now occurring, which means new capabilities are added to a service that might break the implementation of that service and the apps built on it. So a decision has to be made whether to version, that is, to create a new side-by-side implementation. This was recently done on Azure with Azure SQL Database. See the article on what's new in Azure SQL Database version 12 at bit.ly/1Dcjpo4.

The point is that there's a tax to pay in the cloud and you need to be aware of that and factor in the cost of the tax. You have to be connected into the cloud support/change notifications process so you can plan ahead and not be caught off guard. The tax, of course, is change and that cost is higher and more frequent than it is on-premises. Don't let that put you off, though, the upside is much greater. The pace at which you can do things is much faster, the quality will be higher and the risks you take are much lower.

Want to learn more? The AzureCon virtual event lets you hear from Azure experts, view Q&As and find out about all the latest Azure innovations. You can sign up to access all the archived conference content at bit.ly/1KRD76d. ■

Tony Meleg *is a technical product manager in the Microsoft Azure team. He spends his time creating technical content for various audiences, evangelizing Azure and turning complexity into simple concepts.*

Thanks to the following Microsoft technical experts for reviewing this article: *Matt Nunn and Jose Miguel Parrella*

ASP.NET 5 Anywhere with OmniSharp and Yeoman

Shayne Boyer and Sayed Ibrahim Hashimi

As development teams have become more diverse in their tooling choices, frameworks must also provide choice without friction. ASP.NET 5 has embraced cross-platform support, including development through open source tooling such as OmniSharp and hosting in Microsoft Azure using containers like Docker. In this article, we'll show you how you can get started with ASP.NET 5 on the platform of your choice. We'll cover all that you need to begin developing Web applications with ASP.NET 5.

Getting a project up and running can be difficult, as modern Web application development is also riddled with choice. As a Visual Studio user, you might've been almost spoiled by benefiting from the IDE, built-in templates and tooling such as Web Essentials to assist you in getting a new project off the ground. But developers not using Windows and a rich IDE, such as Visual Studio, have generally relied on command-line tooling such as Yeoman, Grunt or Gulp, or Node.js to construct and build Web applications. Now ASP.NET 5 has been rebuilt from the ground up with all platforms in mind, taking the "choice is king" approach for developer tooling. Now you can use it not only for your Windows projects, but also for Linux and OS X. Here's a brief look at setting up and creating a project from a non-Windows OS perspective using ASP.NET 5.

This article discusses:

- Setting up an ASP.NET 5 development environment on OS X
- Creating a project in Yeoman
- Using the Visual Studio Code editor
- Debugging and deploying

Technologies discussed:

ASP.NET 5, Visual Studio Code, Node.js, HomeBrew, OmniSharp, Yeoman

Setting Up Your Environment

You'll need a few pieces to get your environment set up, but the process is well-documented for both OS X and Linux. You'll find step-by-step instructions at bit.ly/1Ljhj68. For this article, we'll assume most of you are using OS X.

The first step is to install the tools we'll use to build our ASP.NET 5 Web application. Eventually, CoreCLR (github.com/dotnet/coreclr) will be the base runtime for the framework. For now, however, ASP.NET 5 still requires the Mono runtime. To install Mono, use HomeBrew (brew.sh):

```
$ brew install mono
```

Next, install the .NET Version Manager (DNVM), a set of command-line utilities that lets you update and configure your .NET execution environment (DNX), which essentially enables cross-platform development using the .NET Core 5 (docs.asp.net/beta5/dnx). To install DNVM and DNX from your terminal, execute the following commands:

```
$ brew tap aspnet/dnx
$ brew update
$ brew install dnvm
```

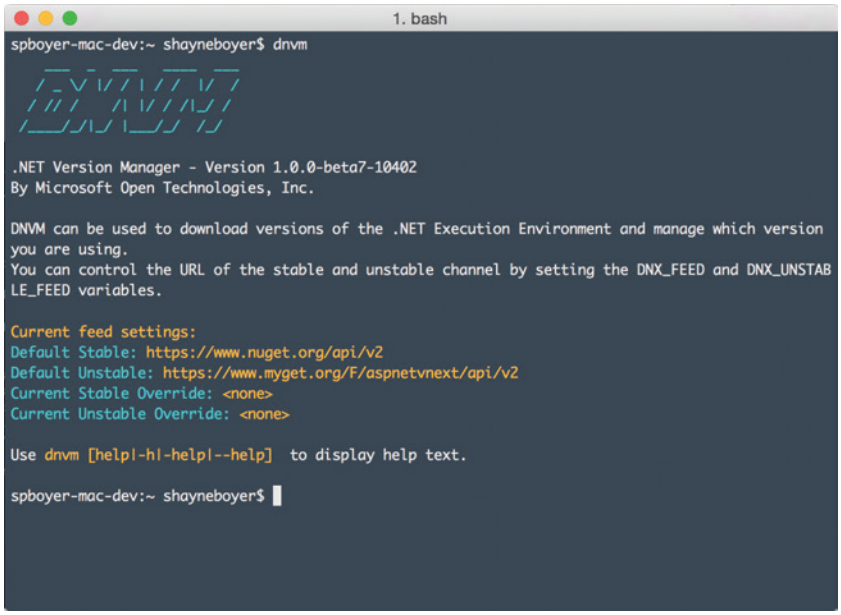


Figure 1 **Checking the DNVM Version**

Now you've installed the Mono runtime, plus DNVM and DNX. To check your DNVM version, type "$ dnvm" at the terminal, as shown in **Figure 1**. Note that if your shell doesn't recognize the dnvm command, you might need to execute "source dnvm.sh" to load it.

Choosing an Editor

If you're using Windows, there's not much debate; you're going to use some version of Visual Studio. However, on OS X or Linux you have a number of choices, from a simple text editor like TextMate to a variety of popular editors such as Sublime, Atom, Emacs or Vim. But there's a new addition to the list of cross-platform development editors—Visual Studio Code (code.visualstudio.com) from Microsoft—and it's our editor of choice not just for ASP.NET 5, but also for AngularJS, Node.js and general JavaScript development (see **Figure 2**).

No matter which tool you decide to use, the key to lighting up the editor for ASP.NET 5 on OS X and Linux is OmniSharp (omnisharp.net). Visual Studio Code comes with OmniSharp built in; other editors have extension or "add-in" repositories where the component can be downloaded.

Starting Your First Project

Without the rich Visual Studio 2015 development environment, you'll have to rely on a different approach for developing an ASP.NET 5 application on OS X. Enter Yeoman (yeoman.io) and the ASP.NET generator project (bit.ly/1MPe5KY). Yeoman is a scaffolding platform built on top of Node.js that allows you to build template-based generators for projects or code files. It's a command-line utility and because it's built on Node.js, there are a few dependencies you'll need to take care of first.

To start, install Node.js and the node package manager (NPM) either via HomeBrew or directly from nodejs.org:

```
$ brew install node
```

When that's complete, install the generators using npm:

```
$ npm install -g yo generator-aspnet
```

If you're not already using Bower, Grunt or Gulp, grab those tools, as well. You'll want to become familiar with these tools as part of the new, modern development stack (see the article "Modern Tools for Web Development: Bower" in this issue):

```
$ npm install -g bower grunt-cli gulp-cli
```

Bower is a package manager for front-end Web development, and a repository for Web resources such as JavaScript and CSS. Grunt and Gulp are task-running libraries for performing build processes such as script and image minification and transpiling (TypeScript or CoffeeScript).

That's it for setting up the tooling you need for development, regardless of the editor. Now, to kick off the new project type, execute "$ yo aspnet" to initialize the Yeoman generator and select the project you'd like to create. In this case, select "Web Application Basic [without Membership and Authorization]." as shown in **Figure 3**, then type your project name and press enter.

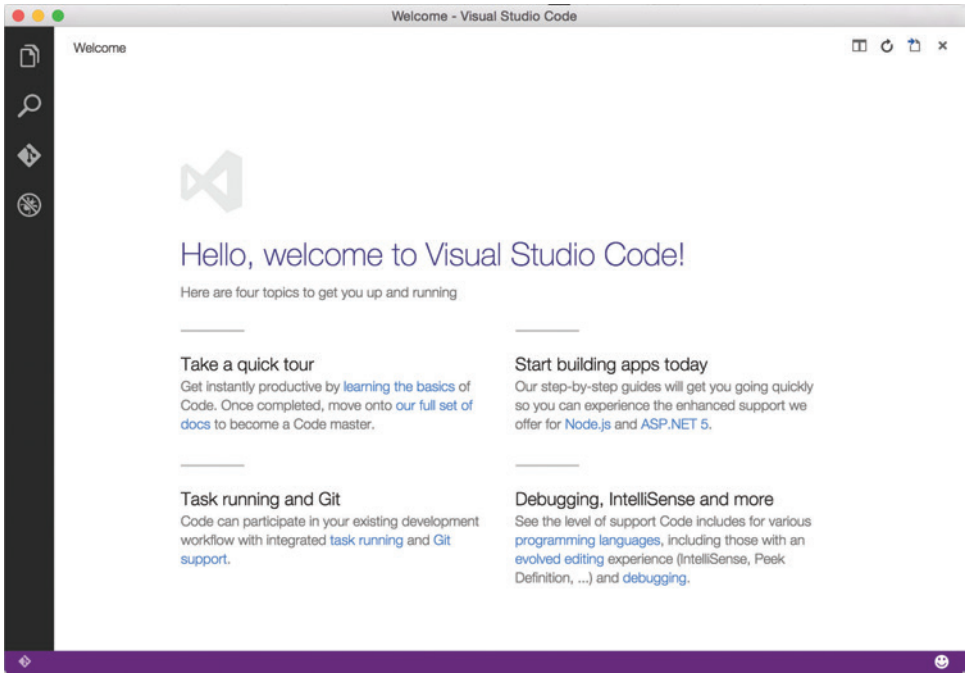


Figure 2 **The Visual Studio Code Welcome Page**

After the generator completes, you have the option of running the application using the Kestrel cross-platform Web server. First, however, you'll need to install the npm, Bower and NuGet dependencies, so run the restore command to get these resources:

```
$ cd [projectname]
$ dnu restore
```

This command downloads all of the NuGet packages for the project referenced in the project.json file.

(We also run "$ npm install" and "$ bower install" to ensure the JavaScript and UI component resources are up-to-date, but this isn't required.)

Next, run the command to start Kestrel:

```
$ dnx . kestrel
```

(Note that after ASP.NET 5 Beta 7 is released, this command will change to simply "dnx kestrel.")

The word "Started" will appear in the terminal window and you'll now be able to view the Web site by browsing to http://localhost:5000. At this point you've created the project, restored packages and run the site without Windows or Visual Studio. Next, you'll open the code in Visual Studio Code.

Editing ASP.NET

As noted earlier, Visual Studio Code is a great editor for cross-platform development. Open the project either the usual way or by using the keyboard shortcut "code" from the project folder. (See bit.ly/1LwonPN for how to set up the "code" shortcut.)

Once you've opened the source folder in Visual Studio Code, you can start to develop the application. **Figure 4** shows the result when you open the project in Code.

As you can see, you get the full syntax highlighting for C# files that you'd expect—in Mac OS X! If you look closely, you'll see a light bulb near the cursor at line 2. The light bulb, just as in Visual Studio, can be used to perform quick actions that are contextual. In this case, Visual Studio Code offers the suggestion to Remove Unnecessary Usings. Now let's add new files to your project.

To add a new file to your ASP.NET 5 project, you don't need to do anything special. Just add a file to the directory and it will automatically be included. In Code you can use the Add File button in the tree view, or Ctrl+N to add a new blank file. If you'd like to get started with some initial content, you can use "yo aspnet." To add files to existing ASP.NET 5 projects, you invoke a sub-generator using the following syntax:

```
$ yo aspnet:<Name> <options>
```

To demonstrate this, let's add a new MVC controller and View for a new Admin page for the Web application. We'll add the MVC controller first. When yo aspnet executes, it will add files to the current working directory, so you'll want to cd into the correct directory before executing the commands. To add the MVC controller, execute the following command in the Controllers folder:

```
yo aspnet:MvcController AdminController
```

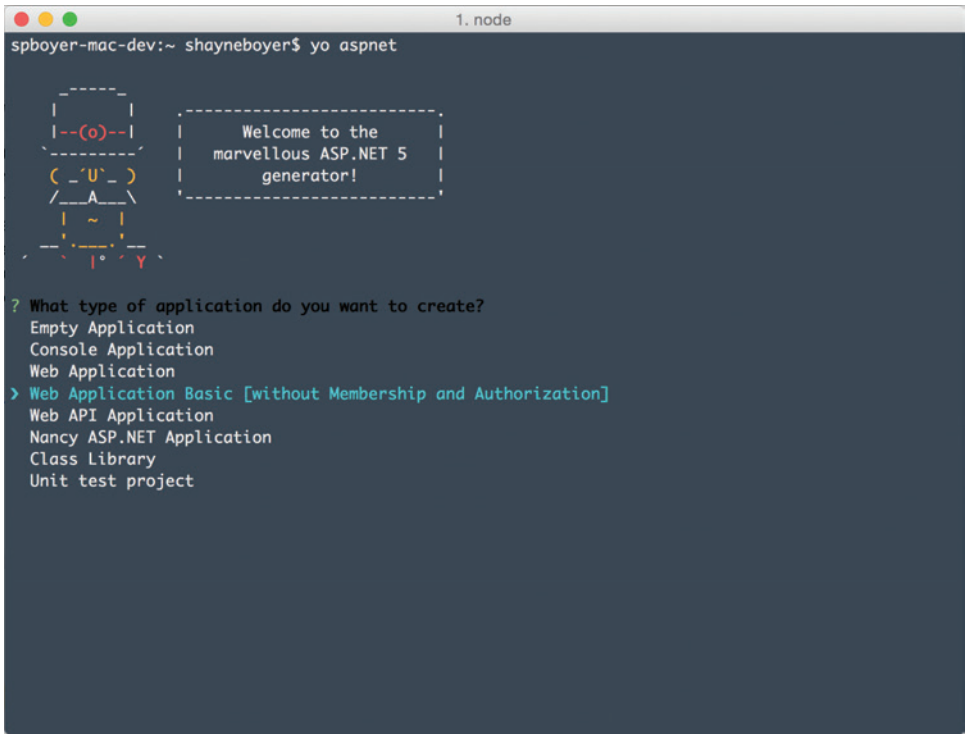


Figure 3 **Selecting the Project Type in Yeoman**

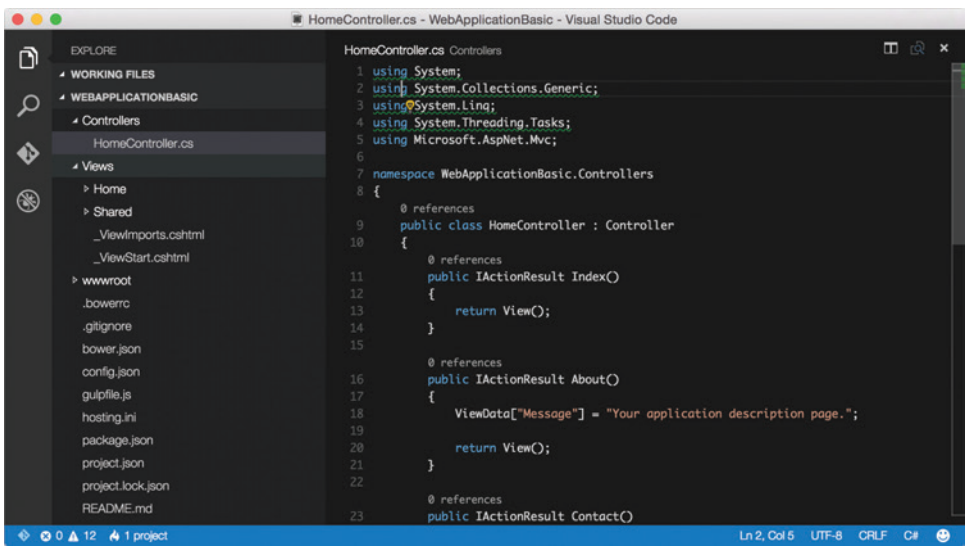


Figure 4 **Opening a Project in the Visual Studio Code Editor**

BEST SELLER

ActiveReports 9 | from $1,567.02

GrapeCity

Award-winning .NET reporting platform for HTML5, WPF, WinForms, ASP.NET & Windows Azure.

- Sophisticated, fast, and powerful reports
- Visual Studio-integrated report designer
- Extensible optional report server with built-in scalability
- Responsive HTML5 Report Portal
- Royalty-free redistribution

BEST SELLER

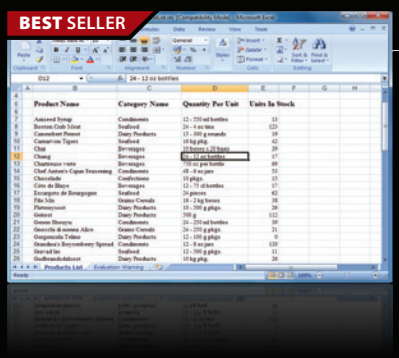
Help & Manual Professional | from $586.04

ec software

Help and documentation for .NET and mobile applications.

- 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 responsive HTML, CHM, PDF, MS Word, ePUB, Kindle or print
- Styles and Templates give you full design control

BEST SELLER



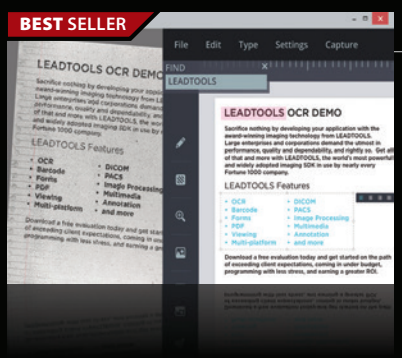
Aspose.Total for .NET | from $2,449.02

ASPOSE
Your File Format APIs

Every Aspose .NET component in one package.

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

BEST SELLER



LEADTOOLS Document Imaging SDKs V19 | from $2,995.00 SRP

LEAD TECHNOLOGIES

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

- Universal document viewer & conversion framework for PDF, Office, CAD, TIFF & more
- OCR, MICR, OMR, ICR and Forms Recognition supporting structured & unstructured forms
- PDF & PDF/A Create, Load, Save, View, Edit
- Barcode Detect, Read, Write for UPC, EAN, Code 128, Data Matrix, QR Code, PDF417
- Zero-footprint HTML5/JavaScript UI Controls & Web Services

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
2 New Century Place
East Street
Reading, Berkshire
RG1 4ET
United Kingdom

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

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

GSA Schedule Contract GS-35F-0188R

Figure 5 **Hello.cs**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNet.Mvc;

// For more information on enabling MVC for empty projects, visit
// go.microsoft.com/fwlink/?LinkID=397860

namespace MyNamespace
{
  public class AdminController : Controller
  {
    // GET: /<controller>/
    public IActionResult Index()
    {
      return View();
    }
  }
}
```

After executing the command, you'll see a new file, Hello.cs, in the current working directory with the content shown in **Figure 5**.

The file looks the same as when you use File | New Item in Visual Studio and select MVC Controller, except that here the namespace name isn't automatically updated; instead, it's hardcoded as MyNamespace. For now you'll need to update the namespace declaration to match what you expect, but this will be updated in a future version. The MvcController sub-generator is just one of many sub-generators available in yo aspnet. To see the full list of sub-generators, you can execute:

```
$ yo aspnet --help
```

> The sub-generators in yo aspnet are equivalent to the item templates in Visual Studio when you use File | Add New Item.

The sub-generators in yo aspnet are equivalent to the item templates in Visual Studio when you use File | Add New Item. To add a view, use the MvcView sub-generator. To add the Admin view, execute the following command from the Views folder:

```
$ yo aspnet:MvcView Index
```

The resulting view, Index.cshtml, is pretty basic:

```
@*
  // For more information on
  // enabling MVC for empty projects,
  // visit bit.ly/1PBdyKc
*@
@{
  // ViewBag.Title = "Index Page";
}
```

Once again, the content generated using yo aspnet is equivalent to the Add New Item dialog in Visual Studio. In the Index.cshtml file you can add a header so you can browse to this page and verify that everything works:

```
<h1>Admin Page</h1>
```

Now let's see what we need to do to build and run this application.

Previously we mentioned that you can use the command "dnx . kestrel" to run your application. If you're using Code you can start the Web server using the command palette shown in **Figure 6**.

When you use Code, your project will build behind the scenes using OmniSharp whenever your source files change. To see any errors and warnings in Code, such as those shown in **Figure 7**, use the errors and warnings button in the status bar. As you can see, Code is indicating bad code at line 16.

You can also build your project from the command line. Let's say your new AdminController class has a build error. To build the application on the command line, execute:

```
$ dnu build
```

This should give you the same errors and warnings that Code shows. Now that you've seen how to build and run your application, let's move on to briefly discuss debugging and deployment.

Debugging

Currently, ASP.NET 5 debugging isn't supported on any platform except Windows and Visual Studio, and that means you can't debug an ASP.NET 5 application running on Mono for OS X or Linux. ASP.NET 5 applications are compiled using the Roslyn compiler, not the Mono compiler, and no debug information is emitted. Visual Studio Code doesn't support debugging yet, but you can always use Visual Studio in a virtual machine on your Mac or Linux machine. Hopefully the Visual Studio Code team will be able to support debugging after the CoreCLR is released.

Deployment

You've learned how to develop your application locally; now let's take a quick look at the hosting options. A detailed examination of this topic would require its own article, so we'll just present a

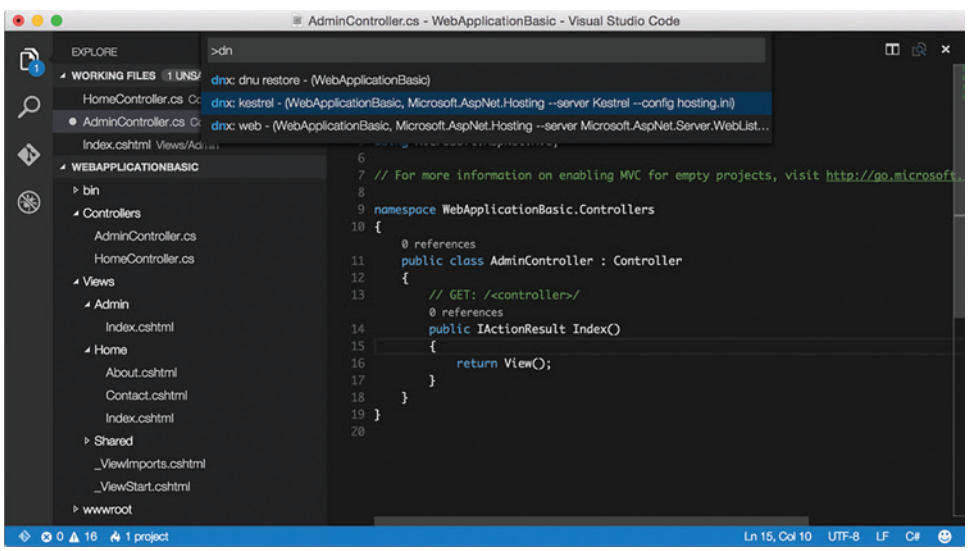


Figure 6 **Starting a Web Server in Visual Studio Code**

high-level overview and point you to some external resources. Visit bit.ly/1fvDQ41 for the latest information.

At a high level, here are the publishing options for ASP.NET 5:

- Command-line publishing using the "dnu publish" command-line utility
- Publishing to Azure Web Apps using Git
- Publishing to a Docker container running in Azure

The line dnu publish command is at the center of each publish method. It will package your application in a format that's runnable on Web servers. Let's take a closer look.

To get started and to to see the available command-line options, execute:

```
dnu publish -help
```

Figure 8 shows the result of executing this command.

The most important command-line option is the --out (-o) argument, which lets you specify the folder to which your files should be published. But you'll also want to explore the other options as needed.

Once you've published the application to a folder, you just need to copy that folder to your Web server. If you're publishing to a Windows machine running IIS, you can configure your Web site just as you always have. For information about how to get your Web server configured on Linux, see bit.ly/1E8uebl.

If you're publishing to Azure, ther's some support you can use to get started. Azure supports ASP.NET 5 applications in Azure Web Apps, as well as in Docker containers. To deploy to Azure Web Apps from a non-Windows machine, you can use either FTP or Git. For the FTP case, you publish the results of dnu publish. See bit.ly/1LnFC2T for more information.

> ASP.NET 5 applications are compiled using the Roslyn compiler, not the Mono compiler, and no debug information is emitted.

The Git-based publish model is easy to use and can support continuous deployment scenarios. To get started publishing to Azure Web Apps using Git, see bit.ly/1hQljSO. That's all you need to know to get started developing and running ASP.NET 5 applications on the platform of your choice.

Wrapping Up

Developing Web applications with ASP.NET used to require that you use Windows and Visual Studio. Now you can use ASP.NET 5 and the related command-line utilities and tools on any platform. And this is just the beginning. To keep an eye on the latest news for ASP.NET 5, visit github.com/aspnet/Home. The yo aspnet project is completely community-driven. If you're interested in helping out, please open an issue at bit.ly/1PvtcGX. ■

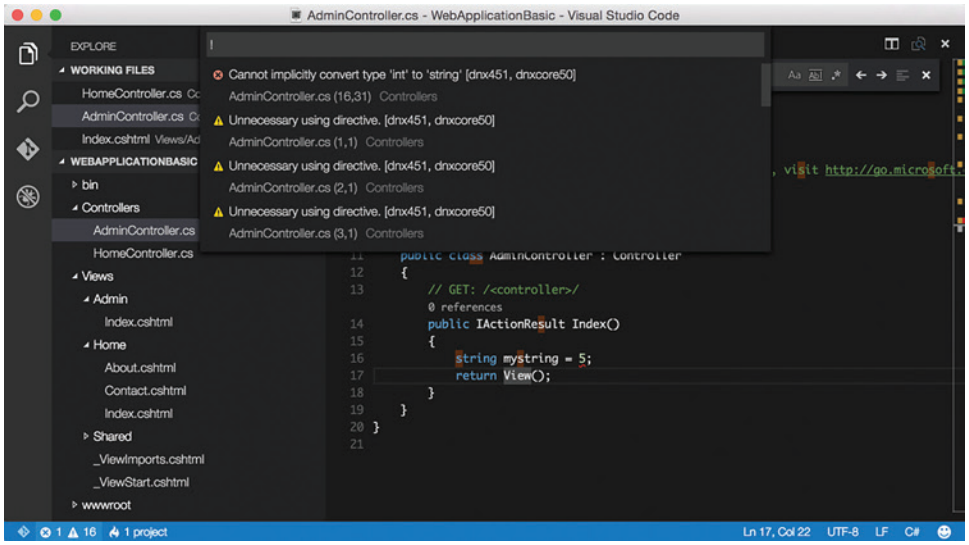


Figure 7 **Viewing Errors and Warnings in Visual Studio Code**

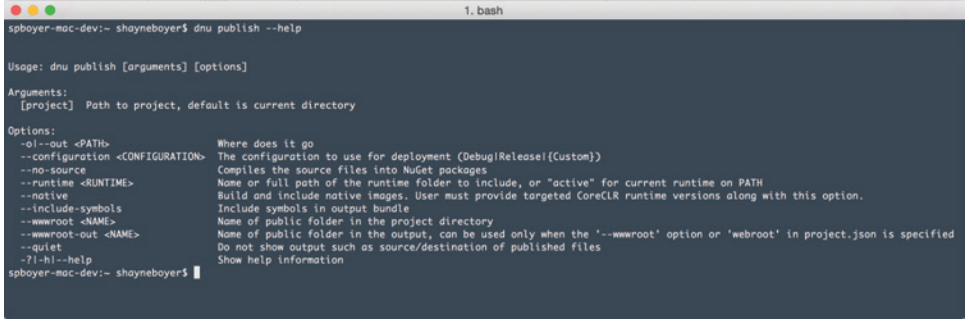


Figure 8 **Getting Help with the "dnu publish" Command**

Shayne Boyer *is an ASP.NET MVP, community speaker and a solutions architect in Orlando, Fla. He has been developing Microsoft-based solutions for the past 20 years. Over the past 10 years, he has worked on large-scale Web applications, with a focus on productivity and performance. You can reach Shayne on Twitter @spboyer and his Web site at tattoocoder.com.*

Sayed Ibrahim Hashimi *is a senior program manager at Microsoft on the Visual Studio Web team. He has written several books on Microsoft technologies and is the creator of SideWaffle and TemplateBuilder, as well as co-creator of OmniSharp. You can reach Sayed on Twitter at @SayedIHashimi and his blog sedodream.com.*

Thanks to the following Microsoft technical expert for reviewing this article: *Scott Hanselman*

NEVRON

www.nevron.com

NEVRON
OPEN VISION for .NET

The Complete UI Suite for .NET

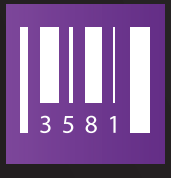
Develop cutting edge User Interfaces from a single code base and embed them in WinForms, WPF, Silverlight, Xamarin.Mac and MonoMac projects.

NOV User Interface for .NET
60+ widgets for rich client side development that are completely free.

NOV Text Editor for .NET
Advanced Microsoft Word-like text editor that can display HTML as well. Suitable for displaying, editing and converting large documents.

NOV Diagram for .NET
Advanced Microsoft Visio-like diagram with full support for smart shapes and many interactivity features.

NOV Map for .NET
Display geographical data from ESRI shape files with build-in data analysis.



NOV Barcode for .NET
Display a large variety of Linear and Matrix barcodes.

NOV Gauge for .NET
Advanced Linear and Radial gauges for display of KPIs and Dashboards.

NOV Chart for .NET
Advanced Charting component for business, presentation, financial, statistical and general use.

NOV Grid for .NET
Advanced hierarchical grid component for displaying and editing relational data.

NOV Scheduler for .NET
Advanced scheduling component for displaying and managing appointments and tasks.

* All NOV components and frameworks are distributes as portable class libraries and work from a single code base inside WinForms, WPF, Silverlight, Xamarin.Mac, MonoMac and soot other environments.

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

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

Modern Tools for Web Development: Bower

Adam Tuliper

For a long, long time, we lived in a beautiful walled garden. In this protected ecosystem of Web development, we used sophisticated technology like ASP.NET and Visual Studio. The rest of the world's tools were considered rather inferior. We were part of an empire, if you will, and it was a pretty good place to be for a long time.

However, as time has passed, development cultures, tools, resources, and more have become fragmented and even chaotic. But some pretty solid tech has cropped up during this period, including Bootstrap, AngularJS, Git, jQuery, Grunt, Gulp, and Bower, and Web developers accustomed to the Microsoft ecosystem are able to take advantage of these tools.

In this first part of a two-article series, I'll provide an overview of Bower, a package manager primarily for (but not limited to) front-end Web development. In the second article, I'll cover Grunt and Gulp, two JavaScript-based task runners that can be used to perform all sorts of tasks, like copying files, minification, concatenation, and even compilation.

This article discusses:

- Installing and using Bower
- Managing dependencies
- Build servers and source control
- Visual Studio support

Technologies discussed:

Bower, Node.js, NuGet, GitHub, ASP.NET 5, Visual Studio 2015

Grunt, Gulp and Bower are additional tools in your Web development arsenal. Tooling to integrate with them is built into Visual Studio 2015 and available via add-ins for Visual Studio 2012 and 2013. You still need to install them.

Some of you might be wondering if Microsoft is making you learn and use even more tools. Doesn't NuGet work just fine for packages, and isn't msbuild sufficient as a build tool? The answer to both questions is yes—in many but not all scenarios. And for those cases where conventional tools don't suffice, Grunt, Gulp and Bower can help. In a Web project, you might want to compile your Sass whenever a CSS file changes. Or you may want to get the latest Bootstrap or Angular release without waiting for someone at Microsoft to create a NuGet package from it. You can't accomplish either task with NuGet or msbuild.

NuGet is an awesome technology and continues to be developed, supported and tightly integrated into Visual Studio. Continue using it for your projects, especially for binaries and projects that need to make changes to your Visual Studio solutions. Each time a new version of jQuery or Bootstrap comes out, though, someone must create and release a NuGet package for it. But because Bower can use semantic versioning, as soon as a tool is released and tagged on GitHub, Bower can use it; no need to wait for someone else to package it up in a NuGet package.

I'll be using the Node Package Manager (npm) for the Bower installation and for several items in the next article. Because npm isn't provided as a standalone download, simply install Node.js from nodejs.org. For more information on installing and using npm,

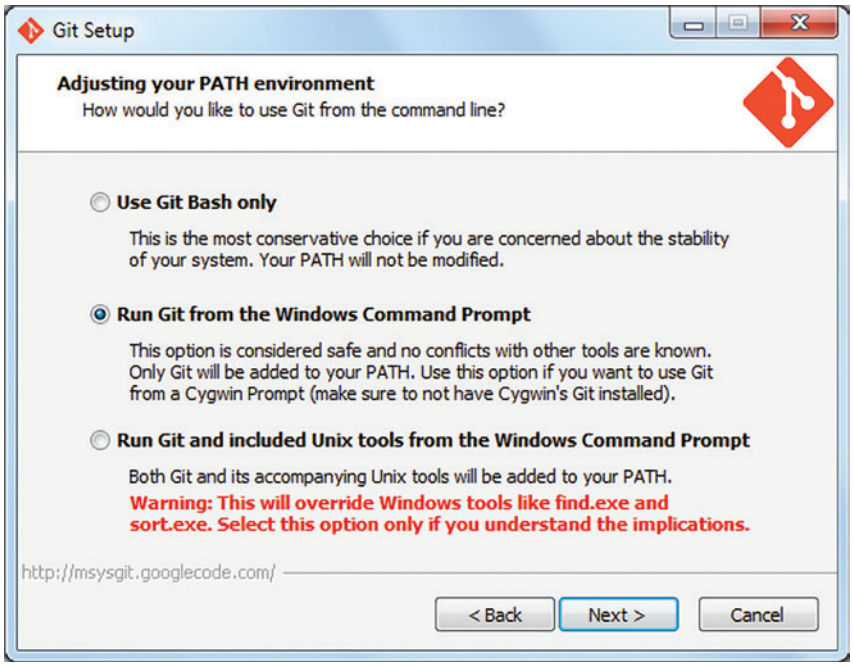


Figure 1 **Install msysgit with Command-Line Support**

visit the Microsoft Virtual Academy "Package Management and Workflow Automation" page at bit.ly/1EjRWMx.

Bower is a major package manager typically used for front-end Web development and, arguably, it's the only front-end-only package manager solution. Most packages used in front-end Web development, such as Bootstrap, jQuery, and AngularJS, can be installed using either npm or Bower, but in many cases the dependency management may be a bit easier with Bower (although some may disagree).

Bower packages, unlike NuGet packages, aren't limited to a single source type. A Bower package can be a Git endpoint, a folder on a file system, a URL for content files or zipped files, and more. Bower integrates with a package registry that lists published packages, but packages don't have to be listed in Bower to be installed.

Installing and Using Bower

Bower typically pulls from a Git repository, so you'll need to install msysgit (msysgit.github.io) and select the option to run from the command prompt, as shown in **Figure 1**.

You use npm to install Bower globally so you can use it from anywhere on your system. You only need to install Bower once, not per project.

```
npm install -g bower
```

Now you're all set to use Bower. Open a command line to your project folder root and use the following format to install a package into your project:

```
bower install <package name/url/zip/etc.> --save
```

For example, to install jquery, simply enter:

```
bower install jquery --save
```

The first three words probably make sense, but the --save might need some explanation. This parameter causes an entry to be written to the bower.json file to note you've installed this package. (Bower doesn't create this file by default; you need to tell it to do so, as I'll discuss shortly.) By default, the Bower install command creates a bower_components folder in the folder where you run the install command; the bower_components folder name can be customized using a Bower configuration file, .bowerrc.

You'll notice in **Figure 2** the jQuery package install results in many more files and folders than you might expect. All I really want in my project is jQuery.js, but in this case I get the entire jQuery source code tree. Many package installs do indeed give you the entire source code tree, often significantly more than you want. This leaves new Bower users wondering which file to use.

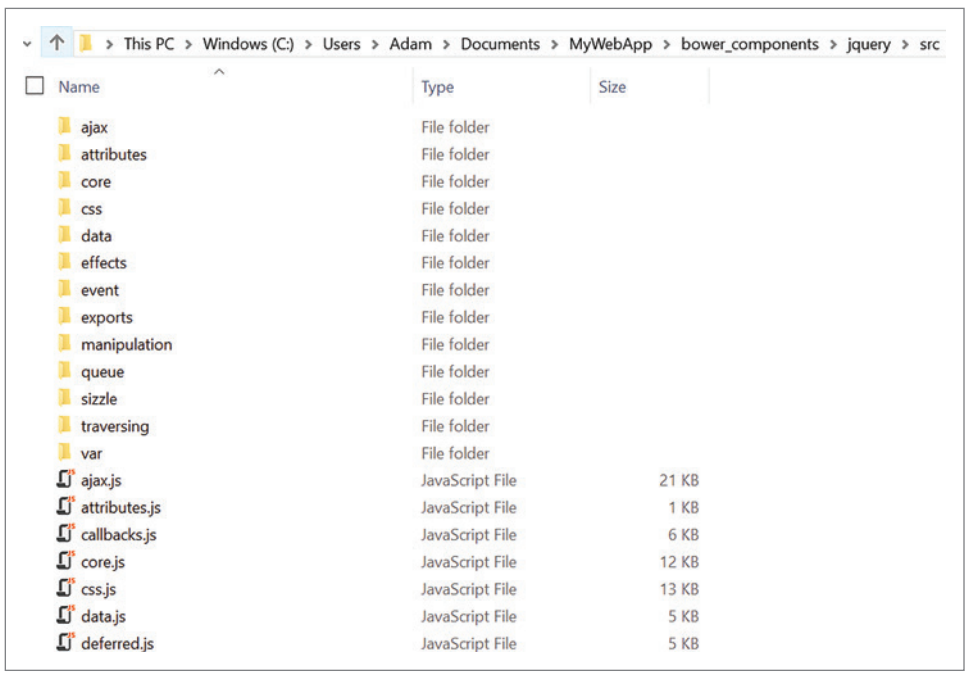


Figure 2 **Bower Installs the Entire jQuery Source Tree**

Some projects will release a packaged version of the app minus all the extra files you don't want. For example, the Bower package for AngularJS is a repository off of the Angular root on GitHub at github.com/angular/bower-angular. When you install this package (bower install angular --save), you get only the .js and .css you need to reference in your HTML pages.

To find packages, you can go to bower.io, use Visual Studio IntelliSense (covered later), or search the Bower package repository via the command line:

```
bower search jquery
```

You can also install several packages at once, which is great when you want to script your installs:

```
bower install jquery bootstrap-css angular
#or install a specific version of jquery ui
bower install jquery-ui#1.10.4
#install a github repository as a package
bower install https://github.com/SomeRepository/project.git
```

Bower creates a local cache of the packages you install. On my Windows 8 and Windows 10 systems, the default cache folder is C:\Users\<MyName>\AppData\Local\bower\cache\packages (you can override this default in your .bowerrc file). As you can see in **Figure 3**, you can manage cached packages via the list and clean commands. Note that Bower will pull from the local cache if it can any time you run bower install.

> Ideally, you want a bower.json file in your application so Bower can track your package dependencies and versions.

For packages that depend on other packages, Bower attempts to install the required dependencies on your file system. Bower has a flat dependency tree, meaning any required dependencies are installed right under /bower_components, not under the package that requires it. For example, the bower.json file for jQuery UI lists a dependency of "jquery": ">=1.6," meaning if jQuery isn't installed yet, the latest version of the jQuery package will be installed to /bower_components, as long as it's at least version 1.6.

Updating or uninstalling packages is pretty straightforward and includes version and dependency checks:

```
#will update based on version rules in bower.json, ex. "jquery": "~2.1.3"
#specifies any patch like 2.1.4 is acceptable to update, but 2.2.0 is not
bower update jquery
#will remove folder and reference in bower.json, but will prompt first
#if other packages have a dependency on jquery
bower uninstall jquery
```

The bower.json File

At this point if I deleted the /bower_components folder, I'd have no idea what packages were installed or required by my application. If I gave the source code (with no packages) to another developer or brought it to another environment, such as a build server without the bower_components folder, that developer and I would be out of luck. If you're familiar with NuGet, this is similar to missing the packages.config file. Ideally, you want a bower.json file in your application so Bower can track your package dependencies and versions, but it's optional.

To create the bower.json file, run the bower init command in your project root and follow the prompts as shown in **Figure 4**.

If you forget to create a bower.json file and install a bunch of packages without it or forget to add those packages using the –save option, don't worry. When you run the bower init command, it asks "set currently installed components as dependencies?" Answering affirmatively means Bower will look at the packages in /bower_components and add what it determines to be the root packages into the dependencies section. Note, however, that if jQuery UI depends on jQuery, jQuery isn't added as a dependency in your file using this method as it's installed when you install jQuery UI. It's always a good idea to review this generated dependencies section to ensure you agree with what dependencies should be listed.

Now you can initialize Bower for your project at the command line, typically in the root folder of your Web project. Then you install your dependencies. A sample bower.json file is shown in **Figure 5**.

The top section contains overall project and package information. Other sections worth mentioning are ignore, dependencies and devDependencies. The ignore section excludes the indicated files if

Figure 3 **Using the Local Bower Cache**

```
#install jquery package for the first time
bower install jquery

#uninstall jquery package
bower uninstall jquery

#install from cache (ie works disconnected)
bower install jquery --offline

#show me the cache
bower cache list

#clean local cache
bower cache clean

#will fail, package no longer cached
bower install jquery --offline
```

Figure 4 **Creating a bower.json File**

```
C:\Users\Adam\Documents\WebApp> bower init
? name: MyWebApp
? version: 1.0.0
? description:
? main file:
? what types of modules does this package expose?
? keywords:
? authors: Adam Tuliper <adam.tuliper@gmail.com>
? license: MIT
? homepage:
? set currently installed components as dependencies? (Y/n) Y
? add commonly ignored files to ignore list? Yes
? would you like to mark this package as private which prevents it from
being accidentally published to the registry? (y/N)
```

Figure 5 **A Sample bower.json File**

```
{
  "name": "MyWebApp",
  "version": "0.0.0",
  "authors": [
    "Adam Tuliper <adam.tuliper@anonymous>"
  ],
  "license": "MIT",
  "ignore": [
    "node_modules",
    "bower_components",
    "test",
    "tests"
  ],
  "dependencies": {
    "angular": "~1.4.3",
    "bootstrap-css": "~3.3.4",
    "jquery-ui": "~1.11.4"
  },
   "devDependencies": {
    "angular-mocks": "~1.4.3"
  }
}
}
```

TEXT CONTROL

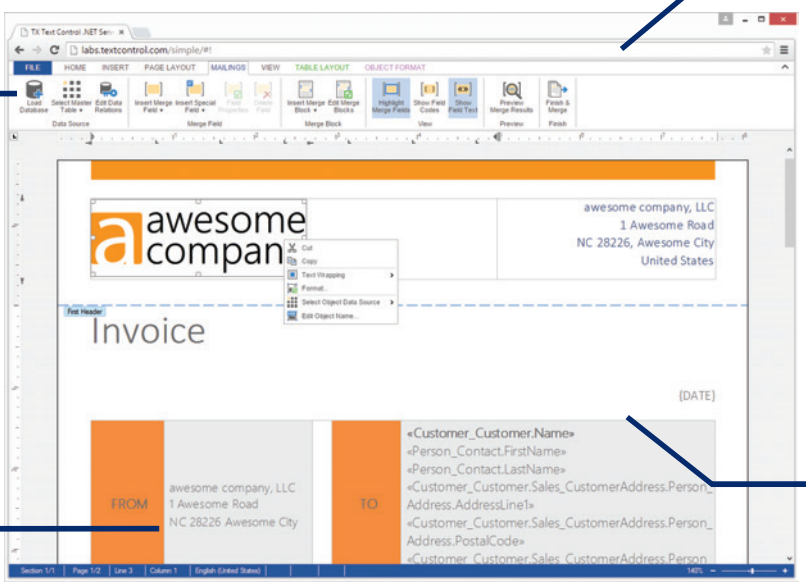
Reporting!

Combine powerful reporting with easy-to-use word processing

The **Text Control Reporting Framework** combines powerful reporting features with an easy-to-use, MS Word compatible word processor. Users create documents and templates using ordinary MS Word skills. TX Text Control is completely independent from MS Word or any other third-party application and can be completely integrated into your business application.

ASP.NET ▪ Windows Forms ▪ WPF

Database support for ADO.NET, ODBC, DataSet, DataTable and all IEnumerable business objects



Cross-browser, cross-platform document and template editing

Create Adobe PDF and PDF/A documents

MS Word compatible templates and MS Word inspired UI

Live demos and 30-day trial version download at:

http://reporting.textcontrol.com

Visual Studio Partner

HTML

TEXT CONTROL

you happen to be creating a Bower package from this app. Because I'm working with a Web app and not creating my own Bower package, this doesn't apply. The dependencies and devDependencies sections contain all of the packages I've installed that will be used by my application; they're covered in the next section in more detail.

Managing Dependencies

You've seen that you can specify two different types of dependencies in a bower.json file: dependencies and devDependencies. Items in the dependencies section are added when you call, for example, bower install jquery --save. These are packages (such as jQuery) that will run in production for your app. The devDependencies entries, on the other hand, are packages that typically won't make it to production, such as mock frameworks like angular-mocks or less/sass compilers. These entries are added when you use, for example, the bower install angular-mocks --save-dev option. These two dependency types are only noted in your bower.json file and don't affect how you use these files on the file system in your application. If you're performing a restore of the packages in, for example, a QA environment, theoretically you wouldn't need to install the devDependencies.

To install all dependencies only in the dependencies section and ignore anything in devDependencies, such as when creating a production build, simply use bower install –production.

If you want to see all of the dependencies your application uses, you can run bower list, which produces output like the following:

```
bower check-new     Checking for new versions of the project dependencies..
MyWebApp#0.0.0 C:\Users\Adam\Documents\MyWebApp
├── angular#1.4.3 (1.4.4-build.4147+sha.bb281f8 available)
├─┬ angular-mocks#1.4.3 (1.4.4-build.4147+sha.bb281f8 available)
│ └── angular#1.4.3 (latest is 1.4.4-build.4147+sha.bb281f8)
├── bootstrap-css#3.3.4
├─┬ jquery-ui#1.11.4
│ └── jquery#2.1.4 (3.0.0-alpha1+compat available)
└── jquery1#2.1.4 (3.0.0-alpha1+compat available)
```

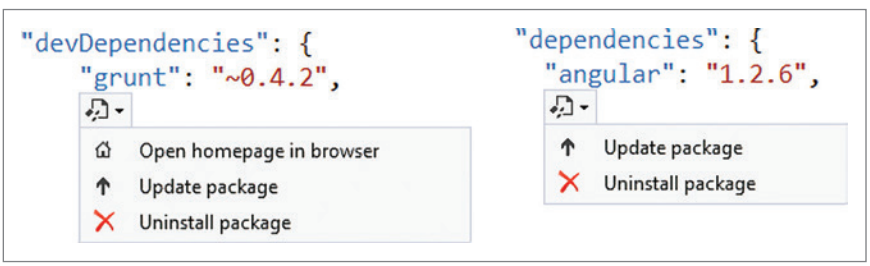


Figure 6 **Package Options in Visual Studio**

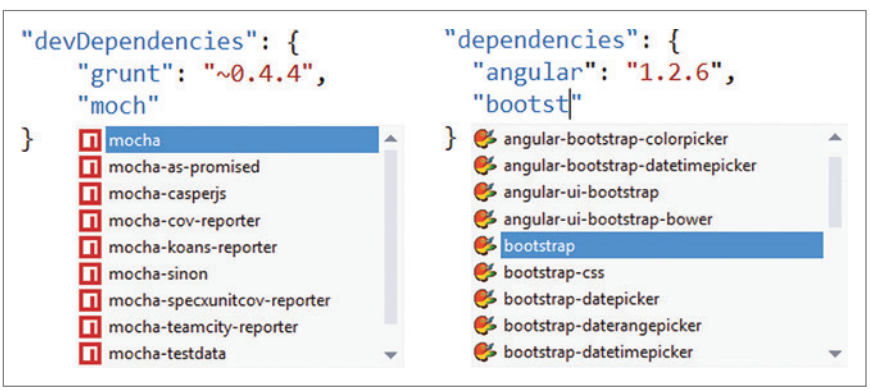


Figure 7 **Bower IntelliSense in Visual Studio**

Too Many Files

Newcomers to Bower quickly realize some packages contain many files while they might need only one file. Some Bower packages list one or more files in the main configuration section that are required for using the package. For example, jQuery has around 90 files in the package when you install it. But to use jQuery, you only need jQuery.js, so the solution is to look at main and note that for jQuery it lists just a single file in the /dist folder:

```
{
  "name": "jquery",
  "version": "2.1.4",
  "main": "dist/jquery.js",
...
}
```

If you look in the /dist folder for jQuery, you'll also find the jQuery.min.js and its corresponding .map file for debugging, although it wouldn't make sense to list these in the main element because the intention is either jQuery.js or jQuery.min.js is used in production, not both.

Running bower list --paths returns all of the main files for every installed package, like so:

```
C:\Users\Adam\Documents\MyWeb> bower list --paths
  angular: 'bower_components/angular/angular.js',
  'bootstrap-css': [
    'bower_components/bootstrap-css/css/bootstrap.min.css',
    'bower_components/bootstrap-css/js/bootstrap.min.js'
  ],
  jquery: 'bower_components/jquery/dist/jquery.js'
```

The onus is on package creators to ensure they list the proper files in the main section.

Because, by default, all package files are under the /bower_components subfolder, you might think my HTML files would reference the files right off this folder as follows:

```
<link rel="stylesheet" type="text/css"
  href="/bower_components/bootstrap-css/css/bootstrap.min.css">
<script src="/bower_components/bootstrap-css/js/bootstrap.min.js" />
<script src="/bower_components/angular/angular.js" />
<script src="/bower_components/jquery/dist/jquery.js" />
```

You can find many examples on the Internet that do this and reference files inside the /bower_components folder. This is not a clean practice. I don't recommend it, and I, for one, don't want to deploy such a folder and potentially hundreds or thousands of files to production when I need only a handful of files that should be minified and concatenated into even fewer files. I'll cover the latter techniques in the next article on Grunt and Gulp, but for now, I'll examine one of several available techniques to pull out these main files in a better way using the bower-installer module.

The bower-installer module will copy all of the main files into your specified folder structure. First, install this module globally on your machine via npm install -g bower-installer.

Next, add a section to your bower.json file to specify where these main files should be copied:

```
"install": {
  "path": "lib"
},
```

Extreme Performance Linear Scalability

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
- Entity Framework & NHibernate Second Level Cache

ASP.NET Optimization in Web Farms

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

Open Source

Full Integration with Microsoft Visual Studio

- NuGet Package for NCache SDK
- Microsoft Certified for Windows Server 2012 R2

Celebrating 10 Years of Market Leadership!

NCache™

FREE Download

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

Your main files for each package will end up in a subfolder under \lib in this case, for example, lib\jquery\jQuery.js, lib\angular\angular.js.

You can customize this process quite a bit. Be sure to check out that package's documentation at bit.ly/1gwKBmZ.

Last, run bower-installer every time you want your files copied to the output folders. Again, there are other ways, such as using Grunt or Gulp to copy these files using a task—you can even watch the folders for changes—but this is a quick, minimal-dependency way of doing it when you start out that doesn't require other workflows.

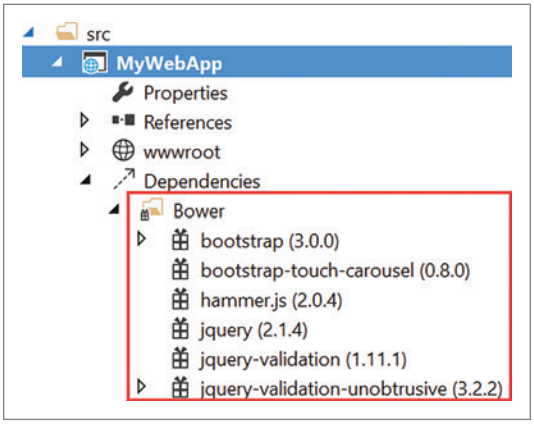


Figure 8 **Bower Dependencies in Solution Explorer**

Build Servers and Source Control

What about when you want to install your required packages on a build server? This is useful in the scenarios where you know only your code is in source control and all other packages (Bower/NuGet, and so forth) are installed and/or built (Sass, Less, CoffeeScript and the like) on a build server. Some shops will source control everything, including all binaries and third-party dependencies. Others rely on the package manager to restore the packages in a build environment. When I give a Visual Studio project to someone, I generally expect the packages to be restored on their machine. The typical recommendation for Bower as it relates to source control, however, is to source control all third-party code if you can live with potentially large repositories or if you don't want to rely on the package manager, otherwise don't source control /bower_components.

To install all dependencies and copy main files using just a bower.json file, simply run the following commands:

```
#Will read bower.json and install referenced packages and their dependencies
bower install

#Optional - copy each packages main{} files into your predefined path
bower-installer
```

Why Not Just npm?

Some developers just use the npm, others choose Bower and some use a mix. You'll find many packages that are in both the Bower and npm package repositories, which makes your workflow easier in any case. The npm works well not only for Node.js apps, but also for managing both client-side and server-side packages. The npm has a nested dependency tree, meaning that for every package you install, all of its dependencies are installed in that package's node_components subfolder. For example, if you use three packages and each one uses jQuery, the entire jQuery package is installed three separate times. Nested dependencies can create quite a long dependency chain. Windows users of the npm are almost guaranteed at some point to run across the dreaded path length error because of this, for example: The directory name C:\Users\Adam\.....\node_modules\somePackageA \node_modules\somePackageB\node_modules\insight\node_modules\inquirer\node_modules\readline2\node_modules\strip-ansi\node_modules\ansi-rege ... is too long.

This is a byproduct of the npm nested dependency structure, though the beta of npm 3 does have a flatten feature. My aim isn't to convince you of one or another as you can happily use both.

Bower, ASP.NET 5 and Visual Studio

Visual Studio 2015 has built-in support for Bower and npm, including installing packages and IntelliSense. Prior versions of Visual Studio can get this same functionality by downloading and installing the free Package IntelliSense Extension for Visual Studio. **Figure 6** shows some of the options that are available to manage your packages inside a bower.json file.

As you can see in **Figure 7**, when you type a package name IntelliSense gives you matching packages and versions, saving you a command-line or Web lookup. Once you make changes to the bower.json file and save it, the package will be installed locally without having to go to the command line. There's nothing here that's Visual Studio-specific from a file standpoint. In other words, a default bower.json from any Web project just works.

Figure 8 shows dependencies from bower.json in Solution Explorer, indicating how Bower is fully integrated into your Web projects and Visual Studio.

With ASP.NET 5, there's a new project structure in which all files in your project folder are included in your project by default, but only items in the /wwwroot folder of your Web project are addressable as static content, such as HTML, CSS, images and JavaScript files. Knowing this, you could set your bower.json config so that bower-installer copies your dependencies to this folder (though the out-of-the-box ASP.NET 5 templates use Gulp to copy pre-set files to their destinations, as I'll cover in the next article).

NuGet packages are still awesome, used heavily and supported, of course, in ASP.NET 5 projects. As a side note, NuGet settings go in the project.json's dependencies section, but show up under references in Visual Studio. NuGet packages are used out-of-the-box for server-side packages, such as logging and MVC support.

Wrapping Up

Bower is a great tool that can easily be integrated into your front-end workflow. Its API is simple to use and with integrated support in Visual Studio, you can use it right along with npm and NuGet to manage both front-end and back-end packages. Take an hour or two to learn Bower; you'll be happy you did. ■

Adam Tuliper *is a senior technical evangelist with Microsoft living in sunny SoCal. He is a Web dev, game dev, Pluralsight author and all-around tech lover. Find him on Twitter @AdamTuliper or adamt@microsoft.com.*

Thanks to the following Microsoft technical expert for reviewing this article: *Michael Palermo*

DocuVieware

NEW Universal HTML5 Viewer & Document Management Kit

- zero-footprint solution
- super-easy integration
- fast & crystal clear rendering
- fully customizable UI look & feel
- mobile devices optimization
- annotations, thumbnails bookmarks & text search

supports nearly 100 formats

Powered by GdPicture.NET 11

FREE TRIAL
www.docuvieware.com

Whether you are an

- Engineer
- Developer
- Programmer
- Software Architect
- Software Designer

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

REGISTER BY OCTOBER 14 AND **SAVE $300!**

Use promo code VSLOCT4 by October 14

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

Take the Tour

LIVE! 360
TECH EVENTS WITH PERSPECTIVE

Visual Studio Live! Orlando is part of Live! 360, the Ultimate Education Destination. This means you'll have access to four (4) other co-located events at no additional cost:

- SharePoint LIVE! — TRAINING FOR COLLABORATION
- SQL Server LIVE! — TRAINING FOR DBAs AND IT PROS
- Modern Apps LIVE! — MOBILE, CROSS-DEVICE & CLOUD DEVELOPMENT
- TECHMENTOR — IN-DEPTH TRAINING FOR IT PROS

Five (5) events and hundreds of sessions to choose from – mix and match sessions to create your own, custom event line-up – it's like no other conference available today!

TURN THE PAGE FOR MORE EVENT DETAILS.

SUPPORTED BY Visual Studio msdn magazine Visual Studio MAGAZINE

PRODUCED BY 1105 MEDIA YOUR GROWTH. OUR BUSINESS.

VSLIVE.COM/ORLANDO

AGENDAS AT-A-GLANCE: VISUAL STUDIO LIVE! & MODERN APPS LIVE!

Cloud Computing	Database and Analytics	Lessons Learned and Advanced Practices	Mobile Client	Visual Studio / .NET Framework	Web Development	Windows Client	Modern Apps Live! Sponsored by: Magenic

START TIME	END TIME	Visual Studio Live! & Modern Apps Live! Pre-Conference: Sunday, November 15, 2015
6:00 PM	9:00 PM	Dine-A-Round Dinner @ Universal CityWalk - 6:00pm - Meet at Conference Registration Desk to walk over with the group

START TIME	END TIME	Visual Studio Live! & Modern Apps Live! Pre-Conference Workshops: Monday, November 16, 2015				
8:00 AM	5:00 PM	**VSM01** Workshop: Service Oriented Technologies: Designing, Developing, & Implementing WCF and the Web API - *Miguel Castro*	**VSM02** Workshop: Triple D: Design, Development, and DevOps - *Billy Hollis and Brian Randell*	**VSM03** Workshop: Busy Developer's Guide to MEANJS - *Ted Neward*	**MAM01** Workshop: Modern App Technology Overview - Android, iOS, Cloud, and Mobile Web - *Nick Landry, Kevin Ford, & Steve Hughes*	
5:00 PM	6:00 PM	EXPO Preview				
6:00 PM	7:00 PM	**Live! 360 Keynote:** Microsoft 3.0: New Strategy, New Relevance - *Pacifica 6* *Mary Jo Foley, Journalist and Author; with Andrew Brust, Senior Director, Datameer*				

START TIME	END TIME	Visual Studio Live! & Modern Apps Live! Day 1: Tuesday, November 17, 2015				
8:00 AM	9:00 AM	**Visual Studio Live! & Modern Apps Live! Keynote:** The Future of Application Development - Visual Studio 2015 and .NET 2015 *Jay Schmelzer, Director of Program Management, Visual Studio Team, Microsoft*				
9:00 AM	9:30 AM	Networking Break • Visit the EXPO - *Pacifica 7*				
9:30 AM	10:45 AM	**VST01** AngularJS 101 - *Deborah Kurata*	**VST02** A Tour of Azure for Developers - *Adam Tuliper*	**VST03** Busy Developer's Guide to NoSQL - *Ted Neward*	**VST04** Visual Studio, TFS, and VSO in 2015 - What's New? -*Brian Randell*	**MAT01** Defining Modern App Development - *Rockford Lhotka*
11:00 AM	12:15 PM	**VST05** From ASP.NET Site to Mobile App in About an Hour - *Ryan J. Salva*	**VST06** Introduction to Next Generation of Azure PaaS – Service Fabric and Containers - *Vishwas Lele*	**VST07** Real World SQL Server Data Tools (SSDT) - *Benjamin Day*	**VST08** Automate Your Builds with Visual Studio Online or Team Foundation Server - *Tiago Pascoal*	**MAT02** Modern App Architecture - *Brent Edwards*
12:15 PM	2:00 PM	Lunch • Visit the EXPO - *Oceana Ballroom / Pacifica 7*				
2:00 PM	3:15 PM	**VST09** I Just Met You, and "This" is Crazy, But Here's My NaN, So Call(Me), Maybe? - *Rachel Appel*	**VST10** Cloud or Not, 10 Reasons Why You Must Know "Websites" - *Vishwas Lele*	**VST11** Windows 10 for Developers: What's New in Universal Apps - *Nick Landry*	**VST12** Defensive Coding Techniques in C# - *Deborah Kurata*	**MAT03** ALM with Visual Studio Online (TFS) and Git - *Brian Randell*
3:15 PM	4:15 PM	Networking Break • Visit the EXPO - *Pacifica 7*				
4:15 PM	5:30 PM	**VST13** Better Unit Tests through Design Patterns for ASP MVC, WebAPI, and AngularJS - *Benjamin Day*	**VST14** Running ASP.NET Cross Platform with Docker - *Adam Tuliper*	**VST15** Build Your First Mobile App in 1 Hour with Microsoft App Studio - *Nick Landry*	**VST16** Putting CodedUI Tests on Steroids - *Donovan Brown*	**MAT04** Reusing Logic Across Platforms - *Kevin Ford*
5:30 PM	7:30 PM	Exhibitor Reception				

START TIME	END TIME	Visual Studio Live! & Modern Apps Live! Day 2: Wednesday, November 18, 2015				
8:00 AM	9:00 AM	**Live! 360 Keynote:** DevOps: What it Means to You - *Pacifica 6 - Sponsored By* PLURALSIGHT *Don Jones, Curriculum Director for IT Pro Content, Pluralsight & Brian Randell, Partner, MCW Technologies*				
9:15 AM	10:30 AM	**VSW01** Mobile App Development with Xamarin and F# - *Rachel Reese*	**VSW02** Notify Your Millions of Users with Notification Hubs - *Matt Milner*	**VSW03** Let's Write a Windows 10 App: A Basic Introduction to Universal Apps - *Billy Hollis*	**VSW04** To Git or Not to Git for Enterprise Development - *Benjamin Day*	**MAW01** Coding for Quality and Maintainability - *Jason Bock*
10:30 AM	11:00 AM	Networking Break • Visit the EXPO - Pacifica 7				
11:00 AM	12:15 PM	**VSW05** Automated UI Testing for Android and iOS Mobile Apps - *James Montemagno*	**VSW06** Busy Developer's Guide to the Clouds - *Ted Neward*	**VSW07** Designing and Building UX for Finding and Visualizing Data in XAML Applications - *Billy Hollis*	**VSW08** Anything C# Can Do, F# Can Do Better - *Rachel Appel & Rachel Reese*	**MAW02** Start Thinking Like a Designer - *Anthony Handley*
12:15 PM	1:45 PM	Birds-of-a-Feather Lunch • Visit the EXPO - *Oceana Ballroom / Pacifica 7*				
1:45 PM	3:00 PM	**VSW09** Stop Creating Forms In Triplicate - Use Xamarin Forms - *Matt Milner*	**VSW10** To Be Announced	**VSW11** Developing Awesome 3D Games with Unity and C# - *Adam Tuliper*	**VSW12** Unit Testing Makes Me Faster: Convincing Your Boss, Your Co-Workers, and Yourself - *Jeremy Clark*	**MAW03** Applied UX: iOS, Android, Windows - *Anthony Handley*
3:00 PM	4:00 PM	Networking Break • Visit the EXPO • Expo Raffle @ 3:30 p.m. - *Pacifica 7*				
4:00 PM	5:15 PM	**VSW13** Go Mobile with C#, Visual Studio, and Xamarin - *James Montemagno*	**VSW14** To Be Announced	**VSW15** Recruiters: The Good, The Bad, & The Ugly - *Miguel Castro*	**VSW16** DI Why? Getting a Grip on Dependency Injection - *Jeremy Clark*	**MAW04** Leveraging Azure Services - *Kevin Ford*
8:00 PM	10:00 PM	Live! 360 Dessert Luau - *Wantilan Pavilion - Sponsored by* amazon web services				

START TIME	END TIME	Visual Studio Live! & Modern Apps Live! Day 3: Thursday, November 19, 2015				
8:00 AM	9:15 AM	**VSH01** Getting Started with ASP.NET 5 - *Scott Allen*	**VSH02** Lessons Learned: Being Agile in a Waterfall World - *Philip Japikse*	**VSH03** Windows, NUI and You - *Brian Randell*	**VSH04** Improving Performance in .NET Applications - *Jason Bock*	**MAH01** Building for the Modern Web with JavaScript Applications - *Allen Conway*
9:30 AM	10:45 AM	**VSH05** Build Data Driven Web Applications with ASP. NET MVC - *Rachel Appel*	**VSH06** User Story Mapping - *Philip Japikse*	**VSH07** Building Adaptive Uis for All Types of Windows - *Ben Dewey*	**VSH08** Asynchronous Tips and Tricks - *Jason Bock*	**MAH02** Building a Modern App with Xamarin - *Nick Landry*
11:00 AM	12:15 PM	**VSH09** Automated Cross Browser Testing of Your Web Applications with Visual Studio CodedUI - *Marcel de Vries*	**VSH10** Performance and Debugging with the Diagnostic Hub in Visual Studio - *Sasha Goldshtein*	**VSH11** XAML Antipatterns - *Ben Dewey*	**VSH12** Roslyn and .NET Code Gems - *Scott Allen*	**MAH03** Building a Modern Cross-Platform App - *Brent Edwards*
12:15 PM	1:30 PM	Lunch on the Lanai - *Lanai / Pacifica 7*				
1:30 PM	2:45 PM	**VSH13** Hate JavaScript? Try TypeScript. - *Ben Hoelting*	**VSH14** Advanced Modern App Architecture Concepts - *Marcel de Vries*	**VSH15** WPF MVVM In Depth - *Brian Noyes*	**VSH16** Getting More Out of Visual Studio Online: Integration and Extensibility - *Tiago Pascoal*	**MAH04** DevOps And Modern Applications - *Dan Nordquist*
3:00 PM	4:15 PM	**VSH17** Grunt, Gulp, Yeoman and Other Tools for Modern Web Development - *Ben Hoelting*	**VSH18** The Vector in Your CPU: Exploiting SIMD for Superscalar Performance - *Sasha Goldshtein*	**VSH19** Building Maintainable and Extensible MVVM WPF Apps with Prism - *Brian Noyes*	**VSH20** Readable Code - *John Papa*	**MAH05** Analyzing Results with Power BI - *Steve Hughes*
4:30 PM	5:45 PM	Live! 360 Conference Wrap-Up - *Andrew Brust (Moderator), Andrew Connell, Don Jones, Rockford Lhotka, Matthew McDermott, Brian Randell, & Greg Shields*				

START TIME	END TIME	Visual Studio Live! & Modern Apps Live! Post-Conference Workshops: Friday, November 20, 2015		
8:00 AM	5:00 PM	**VSF01** Workshop: Angular in 0 to 60 - *John Papa*	**VSF02** Workshop: Native Mobile App Development for iOS, Android and Windows Using C# - *Marcel de Vries & Roy Cornelissen*	**MAF01** Workshop: Modern App Development In-Depth - iOS, Android, Windows, and Web - *Brent Edwards, Anthony Handley, & Allen Conway*

Sessions and speakers subject to change.

Build and Deploy Libraries with Integrated Roslyn Code Analysis to NuGet

Alessandro Del Sole

The Microsoft .NET Compiler Platform (also referred to as the "Roslyn" code base) offers open source C# and Visual Basic compilers that expose, among others, rich code analysis APIs you can leverage to build live analysis rules that integrate into the Visual Studio 2015 code editor. With the .NET Compiler Platform, you can write custom, domain-specific code analyzers and refactorings so Visual Studio can detect code issues as you type, reporting warnings and error messages. A big benefit of the .NET Compiler Platform is that you can bundle code analyzers with your APIs. For instance, if you build libraries or reusable user controls, you can ship analyzers together with your libraries and provide developers an improved coding experience.

In this article, I'll explain how to bundle libraries and analyzers into NuGet packages for online deployment, showing how you can offer integrated Roslyn code analysis for your APIs. This requires you have at least a basic knowledge about .NET Compiler Platform concepts and about writing a code analyzer. These topics have been discussed in past *MSDN Magazine* articles by Alex Turner: "C# and Visual Basic: Use Roslyn to Write a Live Code Analyzer for Your API" (msdn.com/magazine/dn879356) and "C#—Adding a Code Fix to Your Roslyn Analyzer" (msdn.com/magazine/dn904670), which you're strongly encouraged to read before going on here.

This article discusses:

- How to build and deploy libraries
- How to create a Roslyn analyzer that detects code issues while typing, specific for the library's members
- How to bundle both the library and the analyzer into one NuGet package

Technologies discussed:

Microsoft .NET Compiler Platform, Visual Studio 2015, NuGet

Code download available at:

msdn.com/magazine/msdnmag1015

> A big benefit of the .NET Compiler Platform is that you can bundle code analyzers with your APIs.

Preparing a Sample Library to Simulate Custom APIs

The first thing you need is a class library that simulates a custom API. The sample library for this article exposes a simple public method that retrieves common information from an RSS feed, returning a collection of feed items. In Visual Studio 2015, create a new Portable Class Library called FeedLibrary with either C# or Visual Basic and ensure that the minimum target is Windows 8.1, Windows Phone 8.1, and the Microsoft .NET Framework 4.5.1. With this target, the library can also take advantage of the Async/Await pattern with no additional requirements.

Rename the Class1.vb or Class1.cs generated file into FeedItem.vb/.cs. The C# code for this class is in **Figure 1** and the Visual Basic code is in **Figure 2**.

The code is very simple: It downloads the syndicated content from the specified RSS feed's URL, creates an instance of the FeedItem class per feed item, and it finally returns a new collection of items. To use the library, you simply invoke the static ParseFeedAsyncAsync method for C#, as follows:

```
// Replace the argument with a valid URL
var items = await FeedItem.ParseFeedAsyncAsync("http://sampleurl.com/
rss");
```

And for Visual Basic it looks like this:

```
'Replace the argument with a valid URL
Dim items = Await FeedItem.ParseFeedAsyncAsync("http://sampleurl.com/rss")
```

This invocation returns an IEnumerable<FeedItem>, which you can then use according to your needs. Select the Release configuration and build the project; at this point, Visual Studio 2015 generates a library called FeedLibrary.dll, which will be used later.

Figure 1 **Implementing a Class to Retrieve Common Items from an RSS Feed in C#**

```
using System.Net.Http;
using System.Threading.Tasks;
using System.Xml.Linq;

namespace FeedLibrary
{
  // Represent a single content in the RSS feed
  public class FeedItem
  {
    // Properties representing information,
    // which is common to any RSS feed
    public string Title { get; set; }
    public string Author { get; set; }
    public string Description { get; set; }
    public DateTimeOffset PubDate { get; set; }
    public Uri Link { get; set; }

    // Return a collection of FeedItem objects from a RSS feed
    public static async Task<IEnumerable<FeedItem>> ParseFeedAsync(
      string feedUrl)
    {
      var client = new HttpClient();
      // Download the feed content as a string
      var result = await client.GetStringAsync(new Uri(feedUrl));

      // If no result, throw an exception
      if (result == null)
      {
        throw new InvalidOperationException(
          "The specified URL returned a null result");
      }
      else
      {
        // LINQ to XML: Convert the returned string into an XDocument object
        var doc = XDocument.Parse(result);
        var dc = XNamespace.Get("http://purl.org/dc/elements/1.1/");

        // Execute a LINQ query over the XML document
        // and return a collection of FeedItem objects
        var query = (from entry in doc.Descendants("item")
                     select new FeedItem
                     {
                         Title = entry.Element("title").Value,
                         Link = new Uri(entry.Element("link").Value),
                         Author = entry.Element(dc + "creator").Value,
                         Description = entry.Element("description").Value,
                         PubDate = DateTimeOffset.Parse(
                           entry.Element("pubDate").Value,
                     System.Globalization.CultureInfo.InvariantCulture)
                     });

        return query;
      }
    }
  }
}
```

Writing a Roslyn Analyzer

The next step is creating a Roslyn analyzer that provides domain-specific live analysis rules for the custom APIs. The analyzer will detect if the URL supplied as the ParseFeedAsyncAsync method's argument is well-formed, using the Uri.IsWellFormedUriString method; if not, the analyzer will report a warning as you type. Of course, there are plenty of ways to detect if a URL is invalid, but I use this one for the sake of simplicity. In addition, for the same reasons, the analyzer will provide live analysis reporting warnings, but it will not offer any code fixes, which is left to you as an exercise. That said, follow these steps:

1. In Solution Explorer, right-click the solution name, then select Add | New Project.
2. In the Extensibility node of the project templates list, select Analyzer with Code Fix (NuGet + VSIX). Note that the Extensibility node doesn't appear by default. You need to first download the .NET Compiler Platform SDK to fetch the Analyzer with Code Fix template projects. Search Analyzers in the New Project dialog and you'll see the template project to download this SDK.

Figure 2 **Implementing a Class to Retrieve Common Items from an RSS Feed in Visual Basic**

```
Imports System.Net.Http
Imports <xmlns:dc="http://purl.org/dc/elements/1.1/">

'Represent a single content in the RSS feed
Public Class FeedItem

  'Properties representing information
  'which is common to any RSS feed
  Public Property Title As String = String.Empty
  Public Property Author As String = String.Empty
  Public Property Description As String = String.Empty
  Public Property PubDate As DateTimeOffset
  Public Property Link As Uri

  'Return a collection of FeedItem objects from a RSS feed
  Public Shared Async Function ParseFeedAsync(feedUrl As String) As _
    Task(Of IEnumerable(Of FeedItem))
    Dim client As New HttpClient

    'Download the feed content as a string
    Dim result = Await client.GetStringAsync(New Uri(feedUrl, UriKind.Absolute))

    'If no result, throw an exception
    If result Is Nothing Then
       Throw New InvalidOperationException(
         "The specified URL returned a null result")
    Else
      'LINQ to XML: Convert the returned string
      'into an XDocument object
      Dim document = XDocument.Parse(result)

      'Execute a LINQ query over the XML document
      'and return a collection of FeedItem objects
      Dim query = From item In document...<item>
                  Select New FeedItem With {
                  .Title = item.<title>.Value,
                  .Author = item.<dc:creator>.Value,
                  .Description = item.<description>.Value,
                  .PubDate = DateTimeOffset.Parse(item.<pubDate>.Value),
                  .Link = New Uri(item.<link>.Value)}

      Return query
    End If
  End Function
End Class
```

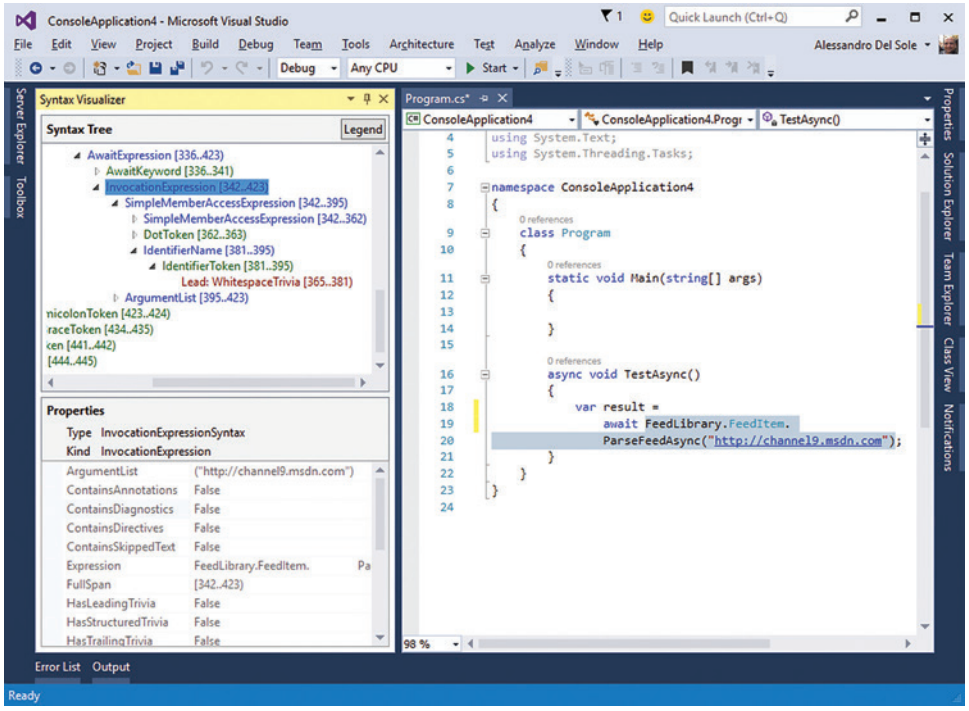


Figure 3 **The Syntax Visualizer Helps Find the Proper Syntax Node Representation**

3. Call the new analyzer FeedLibraryAnalyzer and click OK.
4. When the new project is ready, remove the CodeFixProvider.cs (or .vb) file.

In the DiagnosticAnalyzer.cs (or .vb) file, the first thing to do is supply strings that identify the analyzer in the coding experience. In order to keep the implementation of the analyzer simpler, in the current example I use regular strings instead of the LocalizableString object and resource files, assuming the analyzer will not need to be localized. Rewrite DiagnosticId, Title, Message, Description and Category for C#, as follows:

```
public const string DiagnosticId = "RSS001";
internal static readonly string Title = "RSS URL analysis";
internal static readonly string MessageFormat = "URL is invalid";
internal static readonly string Description =
  "Provides live analysis for the FeedLibrary APIs";
internal const string Category = "Syntax";
```

And for Visual Basic, as follows:

```
Public Const DiagnosticId = "RSS001"
Friend Shared ReadOnly Title As String = "RSS URL analysis"
Friend Shared ReadOnly MessageFormat As String = "URL is invalid"
Friend Shared ReadOnly Description As String =
  "Provides live analysis for the FeedLibrary APIs"
Friend Const Category = "Syntax"
```

Don't change the diagnostic severity, which is Warning by default and is a proper choice for the current example. Now it's time to focus on the analysis logic. The analyzer must check whether the code is invoking a method called ParseFeedAsync. If so, the analyzer will then check if the supplied URL is well-formed. With the help of the Syntax Visualizer, you can see in **Figure 3** how an invocation of the ParseFeedAsync method is represented by an InvocationExpression, mapped to an object of type InvocationExpressionSyntax.

So the analyzer focuses only on objects of type InvocationExpressionSyntax; when it finds one, it converts the associated expression into an object of type MemberAccessExpressionSyntax, which contains information about a method call. If the conversion succeeds, the analyzer checks if the method is ParseFeedAsync, then it retrieves the first argument and performs live analysis on its value. This is accomplished by a new method, called AnalyzeMethod, that works at the SyntaxNode level and is represented in **Figure 4** for C# and **Figure 5** for Visual Basic.

At this point, you need to edit the Initialize method to call the newly added AnalyzeMethodInvocation method, as shown in **Figure 6** for C# and **Figure 7** for Visual Basic.

Notice how the code first checks to see if a reference to the library exists in the project by invoking the Compilation.GetTypeMetadataName method, whose argument is the name of the type that must exist in the current context to make sure that a reference has been added. If this invocation returns null, it means that the type does not exist and, therefore, no reference has been added to the library. So, there's no need to register a code analysis action, improving the analyzer's performances. If you now press F5 to test the analyzer in the Experimental instance of Visual Studio 2015 and create a new project with a reference to the FeedLibrary library, you'll be able to see how it correctly reports a warning every time you supply an invalid URL, as shown in **Figure 8**.

So far you've built APIs and related, domain-specific code analysis rules. Now it's time to see how to bundle both into a single NuGet package.

> If you want to share a library with integrated Roslyn analysis, you need to add the library to the NuGet package that Visual Studio 2015 generates when you build the project.

Building a NuGet Package That Includes APIs and Analyzers

The MSBuild rules for the Analyzer with Code Fix project template automate the generation of a NuGet package that includes the compiled analyzer, which you can share with other developers by publishing it to a NuGet repository. In practice, every time you debug an analyzer by pressing F5 or when you build the analyzer

Spreadsheets Made Easy.

Fastest Calculations

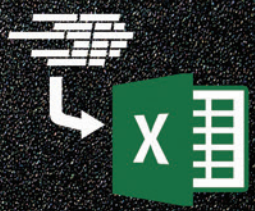
Evaluate complex Excel-based models and business rules with the fastest and most complete Excel-compatible calculation engine available.

Powerful Controls

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

Comprehensive Charting

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



Scalable Reporting

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

Download your free fully functional evaluation at SpreadsheetGear.com

SpreadsheetGear

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

Figure 4 **Detecting Issues on the ParseFeedAsync Argument in C#**

```
private static void AnalyzeMethodInvocation(SyntaxNodeAnalysisContext context)
{
  // Convert the current syntax node into an InvocationExpressionSyntax,
  // which represents a method call
  var invocationExpr = (InvocationExpressionSyntax)context.Node;

  // Convert the associated expression into a MemberAccessExpressionSyntax,
  // which represents a method's information
  // If the expression is not a MemberAccessExpressionSyntax, return
  if (!(invocationExpr.Expression is MemberAccessExpressionSyntax))
  {
    return;
  }
  var memberAccessExpr = (MemberAccessExpressionSyntax)invocationExpr.Expression;

  // If the method name is not ParseFeedAsync, return
  if (memberAccessExpr?.Name.ToString() != "ParseFeedAsync") { return; }

  // If the method name is ParseFeedAsync, check for the symbol
  // info and see if the return type matches
  var memberSymbol = context.SemanticModel.
                     GetSymbolInfo(memberAccessExpr).
                     Symbol as IMethodSymbol;
  if (memberSymbol == null) { return; }
  var result = memberSymbol.ToString();

  if (memberSymbol?.ReturnType.ToString() !=
  "System.Threading.Tasks.Task<
    System.Collections.Generic.IEnumerable<FeedLibrary.FeedItem>>")
  {
      return;
  }

  // Check if the method call has the required argument number
  var argumentList = invocationExpr.ArgumentList;
  if (argumentList?.Arguments.Count != 1) {
      return; }

  // Convert the expression for the first method argument into
  // a LiteralExpressionSyntax. If null, return
  var urlLiteral = (LiteralExpressionSyntax)invocationExpr.ArgumentList.
      Arguments[0].Expression;
  if (urlLiteral == null) { return; }

  // Convert the actual value for the method argument into string
  // If null, return
  var urlLiteralOpt = context.SemanticModel.GetConstantValue(urlLiteral);

  var urlValue = (string)urlLiteralOpt.Value;
  if (urlValue == null) { return; }

  // If the URL is not well-formed, create a diagnostic
  if (Uri.IsWellFormedUriString(urlValue, UriKind.Absolute) == false)
  {
    var diagn = Diagnostic.Create(Rule, urlLiteral.GetLocation(),
      "The specified parameter Is Not a valid RSS feed");
    context.ReportDiagnostic(diagn);
  }
}
```

project, Visual Studio 2015 rebuilds the analyzer .dll file (FeedLibraryAnalyzer.dll in the current example) and a redistributable NuGet package that contains the analyzer.

The build process also generates a VSIX package that you can publish to the Visual Studio Gallery and that's also used to debug the analyzer inside the Experimental instance of Visual Studio 2015, but this is out of scope for this article and isn't covered here.

If you want to share a library with integrated Roslyn analysis, you need to add the library to the NuGet package that Visual Studio 2015 generates when you build the project. Before doing this, you need to understand a bit more about how a NuGet package for an analyzer is made. Actually, a NuGet package is a .zip archive with .nupkg extension. Because of this, you can easily investigate the contents and structure of a NuGet package with a .zip archiver tool, such as the Windows Explorer compressed folder tools, WinZip or WinRar. Following is a summary of the most important items in a NuGet package designed to deploy analyzers:

- **.nuspec file:** This file contains package metadata and includes information required for publication, such as package name, version, description, author, license URL and more. The .nuspec file is bundled into the NuGet package based on the Diagnostic.nuspec file that you see in Solution Explorer, within the analyzer project. You'll edit Diagnostic.nuspec in Visual Studio 2015 shortly.

Figure 5 **Detecting Issues on the ParseFeedAsync Argument in Visual Basic**

```
Private Sub Shared AnalyzeMethodInvocation(context As SyntaxNodeAnalysisContext)
  'Convert the current syntax node into an InvocationExpressionSyntax
  'which represents a method call
  Dim invocationExpr = CType(context.Node, InvocationExpressionSyntax)

  'Convert the associated expression into a MemberAccessExpressionSyntax
  'which represents a method's information
  'If the expression Is Not a MemberAccessExpressionSyntax, return
  If TypeOf invocationExpr.Expression IsNot MemberAccessExpressionSyntax Then Return
  Dim memberAccessExpr = DirectCast(invocationExpr.Expression,
    MemberAccessExpressionSyntax)

  'If the method name Is Not ParseFeedAsync, return
  If memberAccessExpr?.Name.ToString <> "ParseFeedAsync" Then Return

  'If the method name is ParseFeedAsync, check for the symbol info
  'and see if the return type matches
  Dim memberSymbol = TryCast(context.SemanticModel.
      GetSymbolInfo(memberAccessExpr).Symbol, IMethodSymbol)
  If memberSymbol Is Nothing Then Return
  Dim result = memberSymbol.ToString

  If Not memberSymbol?.ReturnType.ToString =
    "System.Threading.Tasks.Task(Of System.Collections.Generic.IEnumerable(
    Of FeedLibrary.FeedItem))"
    Then Return

  'Check if the method call has the required argument number
  Dim argumentList = invocationExpr.ArgumentList
  If argumentList?.Arguments.Count <> 1 Then Return

  'Convert the expression for the first method argument into
  'a LiteralExpressionSyntax. If null, return
  Dim urlLiteral =
    DirectCast(invocationExpr.ArgumentList.Arguments(0).GetExpression,
      LiteralExpressionSyntax)
  If urlLiteral Is Nothing Then Return

  'Convert the actual value for the method argument into string
  'If null, return
  Dim urlLiteralOpt = context.SemanticModel.GetConstantValue(urlLiteral)

  Dim urlValue = DirectCast(urlLiteralOpt.Value, String)
  If urlValue Is Nothing Then Return

  'If the URL Is Not well-formed, create a diagnostic
  If Uri.IsWellFormedUriString(urlValue, UriKind.Absolute) = False Then
     Dim diagn = Diagnostic.Create(Rule, urlLiteral.GetLocation,
       "The specified parameter Is Not a valid RSS feed")
     context.ReportDiagnostic(diagn)
  End If
End Sub
```

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

Figure 6 **Editing the Initialize Method in C#**

```
public override void Initialize(AnalysisContext context)
{
  // Register an action when compilation starts
  context.
    RegisterCompilationStartAction((CompilationStartAnalysisContext ctx) =>
  {
    // Detect if the type metadata
    // exists in the compilation context
    var myLibraryType =
    ctx.Compilation.
    GetTypeByMetadataName("FeedLibrary.FeedItem");

    // If not, return
    if (myLibraryType == null)
        return;

    // Register an action against an InvocationExpression
    ctx.RegisterSyntaxNodeAction(AnalyzeMethodInvocation,
      SyntaxKind.InvocationExpression);
  });
}
```

Figure 7 **Editing the Initialize Method in Visual Basic**

```
Public Overrides Sub Initialize(context As AnalysisContext)
  ' Register an action when compilation starts
  context.
    RegisterCompilationStartAction
    (Sub(ctx As CompilationStartAnalysisContext)
      'Detect if the type metadata
      'exists in the compilation context
      Dim myLibraryType =
        ctx.Compilation.
          GetTypeByMetadataName("FeedLibrary.FeedItem")
        'If not, return
        '(no reference to the library)
        If myLibraryType Is Nothing
Then Return
        'Register an action against
        'an InvocationExpression
        ctx.RegisterSyntaxNodeAction(
          AddressOf AnalyzeMethodInvocation,
          SyntaxKind.InvocationExpression)
     End Sub)
End Sub
```

- **tools folder:** This folder contains Windows PowerShell scripts used by Visual Studio to install (Install.ps1) and uninstall (Uninstall.ps1) an analyzer for a given project.
- **analyzers folder:** This folder contains analyzer .dll files organized into particular subfolders. Agnostic analyzer libraries (that is, targeting all languages) reside in a subfolder called dotnet. Analyzers targeting C# reside in a subfolder called dotnet\cs, whereas analyzers targeting Visual Basic reside in a folder called dotnet\vb. It's worth mentioning that dotnet represents the NuGet profile for .NET Core, and supports projects types such as Universal Windows apps and ASP.NET 5 projects.

A number of additional items can be bundled into a NuGet package, but here I focus on a typical package generated for a Roslyn analyzer, so only the required items are discussed.

Any libraries that can be referenced from a Visual Studio project must be organized into a folder called lib. Because libraries can target different platforms, such as different versions of the .NET Framework, the Windows Runtime, different versions of Windows Phone, or even a portable subset (including Xamarin libraries), the lib folder must contain one subfolder per targeted platform, and each subfolder must contain a copy of the library to deploy. The name of each subfolder must match the name of a so-called profile representing a specific platform. For instance, if you have a library targeting the .NET Framework 4.5.1 and Windows 8.1, you'd have the following structure, where net451 is the profile name for the .NET Framework 4.5.1 and netcore451 is the profile name for the Windows Runtime in Windows 8.1:

```
lib\net451\mylib.dll
lib\netcore451\mylib.dll
```

It is worth mentioning the uap10.0 profile that targets the Universal Windows Platform (UWP) to build Windows 10 apps. The full list of supported profiles is available in the NuGet documentation.The sample library created previously is a portable library targeting .NET Framework 4.5.1, Windows 8.1 and Windows Phone 8.1. The profile name for this kind of target is portable-net451+netcore451+wpa81 and must be used to name the subfolder that will contain the library in the NuGet package. You don't need to create the subfolder and copy the library manually; you simply need to edit the NuGet package metadata (the Diagnostic.nuspec file) inside Visual Studio. **Figure 9** shows updated metadata with proper information for publishing (id, title, author, description, license and so on) and a new file element in the files node that specifies the source file and the target subfolder.

The src attribute indicates the source location for the library, and target specifies the target subfolder, based on the proper profile name. In this case, Visual Studio will search for a library called FeedLibrary.dll

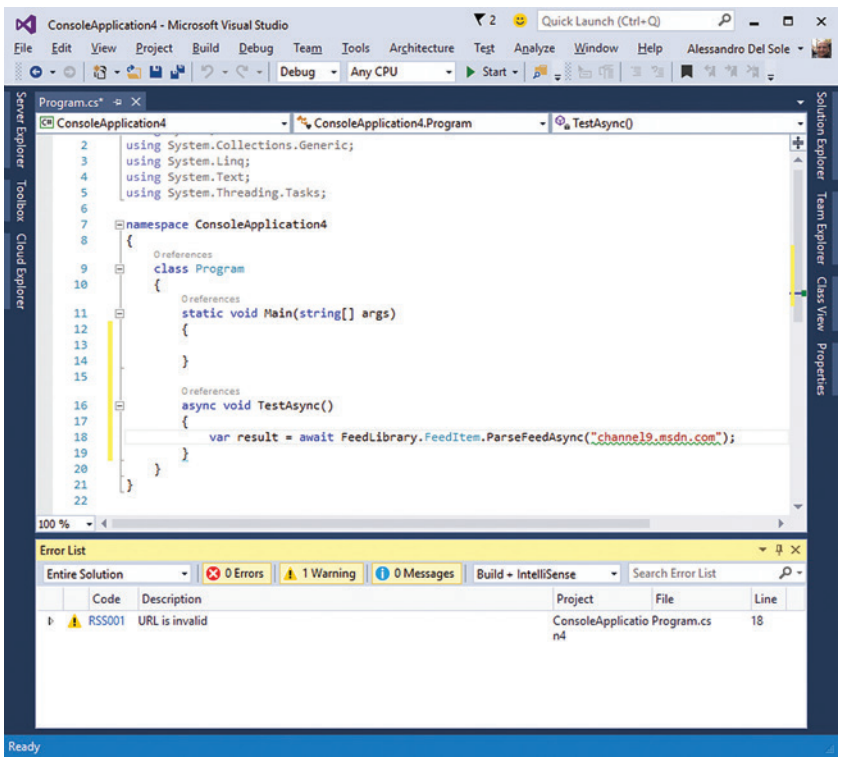


Figure 8 **The Analyzer Reports a Warning If the URL Is Not Well-Formed**

Figure 9 **Editing the NuGet Package Metadata**

```
<?xml version="1.0"?>
<package xmlns="http://schemas.microsoft.com/packaging/2011/08/nuspec.xsd">
  <metadata>
    <id>RSSFeedLibrary</id>
    <version>1.0.0.0</version>
    <title>RSS Feed Library with Roslyn analysis</title>
    <authors>Alessandro Del Sole</authors>
    <owners>Alessandro Del Sole</owners>
    <licenseUrl>http://opensource.org/licenses/MIT</licenseUrl>
    <!-- Removing these lines as they are not needed
    <projectUrl>http://PROJECT_URL_HERE_OR_DELETE_THIS_LINE</projectUrl>
    <iconUrl>http://ICON_URL_HERE_OR_DELETE_THIS_LINE</iconUrl>-->
    <requireLicenseAcceptance>true</requireLicenseAcceptance>
    <description>Companion sample for the "Build and Deploy Libraries
    with Integrated Roslyn Code Analysis to NuGet" article
    on MSDN Magazine</description>
    <releaseNotes>First release.</releaseNotes>
    <copyright>Copyright 2015, Alessandro Del Sole</copyright>
    <tags>RSS, analyzers, Roslyn</tags>
    <frameworkAssemblies>
      <frameworkAssembly assemblyName="System" targetFramework="" />
    </frameworkAssemblies>
  </metadata>
  <files>
    <file src="*.dll" target="analyzers\dotnet\cs"
          exclude="**\Microsoft.CodeAnalysis.*;
          **\System.Collections.Immutable.*;**\System.Reflection.Metadata.*;
          **\System.Composition.*" />
    <file src="tools\*.ps1" target="tools\" />
    <file src="lib\FeedLibrary.dll" target="lib\portable-net451+netcore451+wpa81\"/>
  </files>
</package>
```

inside a folder called lib, which you have to create in the current directory. The latter is the folder that contains the compiled analyzer, typically the Release or Debug folder, depending on the selected build configuration. Based on the current example, you'll need to create a folder called lib inside the Release folder, then copy the FeedLibrary.dll (generated when compiling the sample library at the beginning) into the lib folder. Once you've done this, you can simply rebuild your solution and Visual Studio will generate an updated NuGet package containing both the library and the Roslyn analyzer.

It is worth noting that, when you rebuild the project, Visual Studio 2015 automatically updates the build and revision version numbers of the NuGet package, disregarding the numbers supplied in the version tag. The updated content of the NuGet package can be easily investigated by opening the updated, highest version of the NuGet package with a .zip archiver tool. Remember: Every time you change the id element's value, as in **Figure 9**, Visual Studio 2015 generates a NuGet package with a different name, based on the new id. In this case, changing the id value with RSSFeedLibrary results in a NuGet package called RSSFeedLibrary.1.0.xxx.yyy.NuPkg, where xxx is the build version number and yyy is the revision version number, both automatically supplied at build time. At this point, you've achieved the first objective: packaging custom APIs with integrated Roslyn analysis into one NuGet package.

As an alternative, you could create (and publish) two separate packages—one for the analyzer and one for the library—and a third empty NuGet package that pulls them together by resolving them as dependencies. With this approach, you can decide to keep the installation size smaller by using the APIs only without the analyzer, though you'll need to be familiar with NuGet conventions to manually create a package from scratch. Before publishing the newly generated package to the online NuGet repository, it's a good idea to test it locally.

Testing a NuGet Package Locally

Visual Studio 2015 allows picking NuGet packages from local repositories. This is very useful, especially in situations in which you need to cache packages you use often or that you might need when working offline. Notice that this is a good solution when the number of packages in the local folder is small; if you had hundreds of packages, it would be much more complex to handle. In order to test the NuGet package generated previously, create a new folder on disk called LocalPackages, then copy the latest version of the RSSFeedLibrary.1.0.xxx.yyy.nupkg file into this folder. The next step is to enable Visual Studio 2015 to pick packages from the specified local folder, as shown in **Figure 10**; select Tools | Options | NuGet Package Manager | Package Sources and, in the Available package sources box, click the Add button (the addition symbol in green). At this point, in the Name and Source text boxes type Local packages and C:\LocalPackages, respectively. Finally, click Update to refresh the list of package sources.

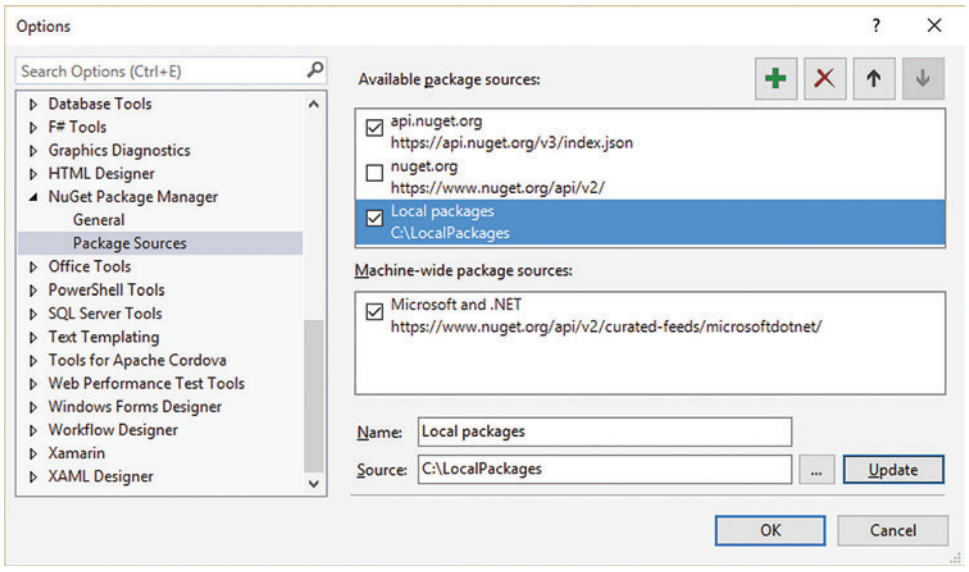


Figure 10 **Adding a Local Repository as a NuGet Package Source**

Now that you have your local NuGet repository, in Visual Studio 2015 create a new console application to test the library plus Roslyn analysis package. Save the project, then select Project | Manage NuGet Packages. When the NuGet Package Manager window appears, in the Package source combo box select the Local packages source. At this point, the package manager will show a list of packages available inside the specified repository, in this case only the sample package.

Click Install. As with NuGet packages online, Visual Studio will show summary information for the selected package and will ask you to accept the license agreement. When the installation completes, you'll be able to see both the library and the Roslyn analyzer in Solution Explorer, as shown in **Figure 11**.

If you simply write some code that uses the FeedItem.ParseFeedAsync method to pass an invalid URL as the argument, the live analysis engine will report a warning message, as expected (see **Figure 8** for reference).

When you install an analyzer, no matter if it was produced by you or by other developers, you can look at details of each rule in Solution Explorer by expanding References, Analyzers, and then the name of the analyzer. In this case, you can expand the name of FeedLibraryAnalyzer and see the RSS URL analysis rule, as shown in **Figure 11**. When you click a rule, the Properties window shows detailed information such as the default and effective severity, if it's enabled by default, and the full rule description. Additionally, you can use the Ruleset Editor to view all the rules applicable to a project and to view or change a rule's severity, as well as disabling or enabling analyzers and rules. To open the Ruleset Editor, in Solution Explorer double-click Properties, then in the project's Properties window select the Code Analysis tab, finally click Open, leaving unchanged the default rule set.

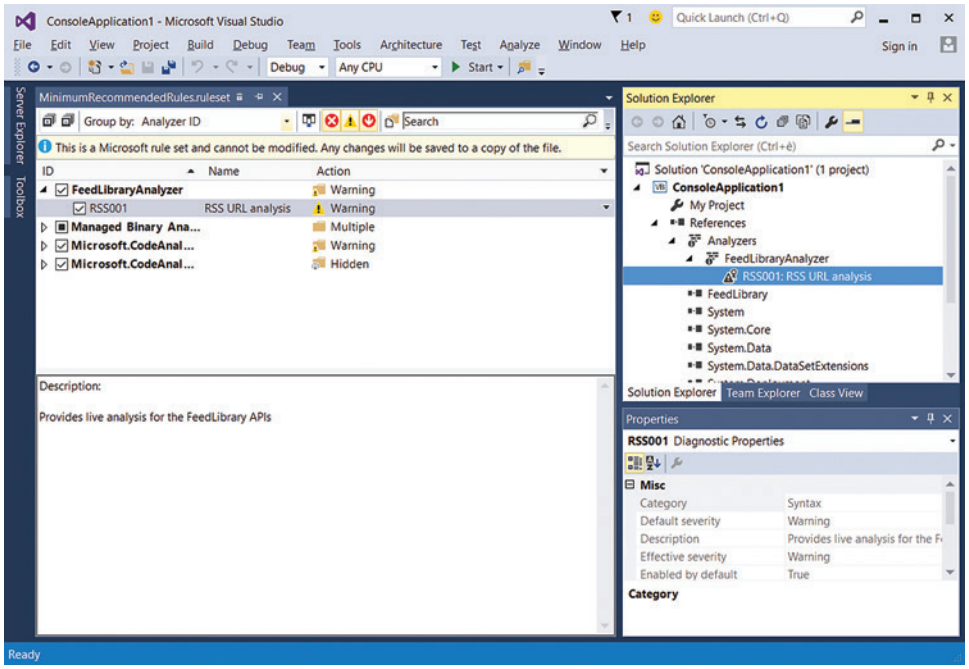


Figure 11 **Both the Library and the Roslyn Analyzer Have Been Installed Via the NuGet Package**

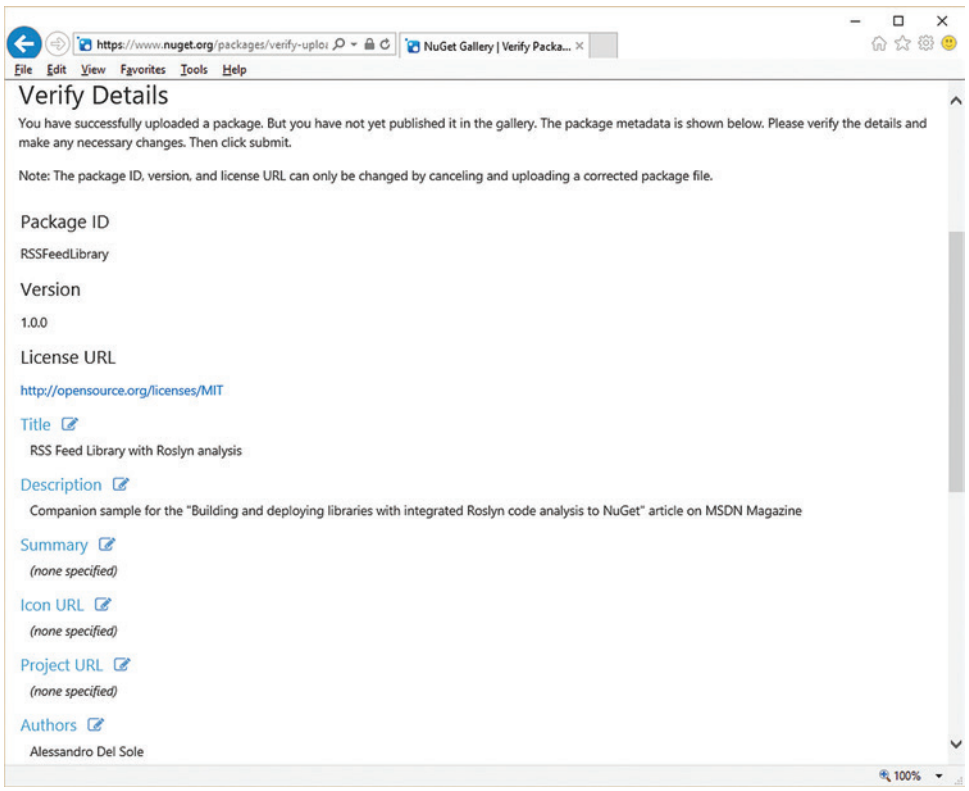


Figure 12 **Reviewing Package Details Before Publication**

As you can see from **Figure 11**, disabling/enabling a rule can be done by deselecting/selecting the checkbox near the rule code; you can change the default severity by clicking the black down arrow on the right side of the current severity level (this can also be done by right-clicking the rule in Solution Explorer and then selecting Set Rule Set Severity from the context menu). When you're satisfied with your local tests, you can move on to publishing the NuGet package online.

Testing a Package Online

On NuGet, developers expect to find high-quality, professional packages. For this reason, before you publish a package to the online NuGet gallery, you should test your work with the help of an online service that allows creating private NuGet repositories and feeds, and graduate to the official NuGet repository once your package is stable. A good choice is offered by MyGet (myget.org), an online service that allows creating personal and enterprise NuGet

feeds, plus VSIX, npm and Bower feeds. MyGet offers a free plan with the most common features required to publish and consume NuGet packages; the free plan does not allow you to create private feeds (you need a paid plan for that), but it is a great choice to test if your packages work as expected from an online repository. When you register, you have an option to create a NuGet feed. For instance, my public feed on MyGet is available at myget.org/F/aledelsole/api/v2. Explaining how to work with MyGet is out of the scope of this article, but the documentation fully describes how to configure your NuGet feed. Once you've created a feed and published your package, you simply enable Visual Studio 2015 to pick NuGet packages from the MyGet feed. To accomplish this, you can follow the steps described in the previous section and take **Figure 10** as a reference, supplying your MyGet feed's URL. To download and test a package in Visual Studio, you will still follow the steps described in the previous section, obviously selecting the MyGet feed as the source in the NuGet Package Manager window.

Publishing a Package to the Online NuGet Gallery

In order to publish packages to the online NuGet repository, you need to open nuget.org and sign in with an account. If you don't have an account yet, click the Register/Sign In hyperlink at the upper-right corner of the page. You can sign in either with a Microsoft Account (recommended) or with username/password credentials. Once you've registered, click Upload Package. The first thing you'll be asked is to specify the NuGet package you want to upload, so click Browse, select the latest version of the RSSFeedLibrary.1.0.xxx.yyy.nupkg from disk, and then click Upload.

Now you'll be prompted with the metadata information for the package. Here you have an option to review the package details before publishing it to the gallery, as shown in **Figure 12**. When ready, click Submit. At this point, the package containing both your APIs and the integrated Roslyn analyzer will be published to the NuGet gallery. Notice that it usually takes 15 to 20 minutes before you can see the package available in the NuGet Package Manager window in Visual Studio 2015.

After the package is listed in the gallery, you'll be able to install it into your project from the online NuGet repository, as shown in **Figure 13**. Once installed, you'll use the library as explained in the previous section, with the integrated live, domain-specific code analysis powered by the .NET Compiler Platform.

Package Updates

As for any NuGet packages, you can improve your libraries and push an updated package version to NuGet. To create an updated package, simply rebuild your solution and Visual Studio 2015 will automatically update the package version number. Then rebuild the project and repeat the steps previously described to publish the NuGet package online. NuGet will handle a list of available versions for each package and will allow developers to choose the version they need.

Wrapping Up

One of the biggest benefits offered by the .NET Compiler Platform is that you can create domain-specific code analysis rules for your APIs. In this article, I first showed how to create a sample library, then how to create a Roslyn analyzer that detects code issues while typing, specific for the library's members. Next, I showed how to bundle both the library and the analyzer into one NuGet package, which is the core of the article. In this way, developers that download the NuGet package will get the APIs with integrated Roslyn live analysis. I then moved on to discuss how to test the NuGet package locally, before publishing online. This step is very important because you have an opportunity to verify that the library/analyzer pair works properly before you make it available to the public. But the fun is when other developers can use the result of your work, so in the last section of the article I showed how to publish the package to the online NuGet repository, and how it can be later installed into your projects from Visual Studio. Deploying Roslyn analyzers alongside your libraries definitely increases the value of your work, providing a better coding experience to other developers. ■

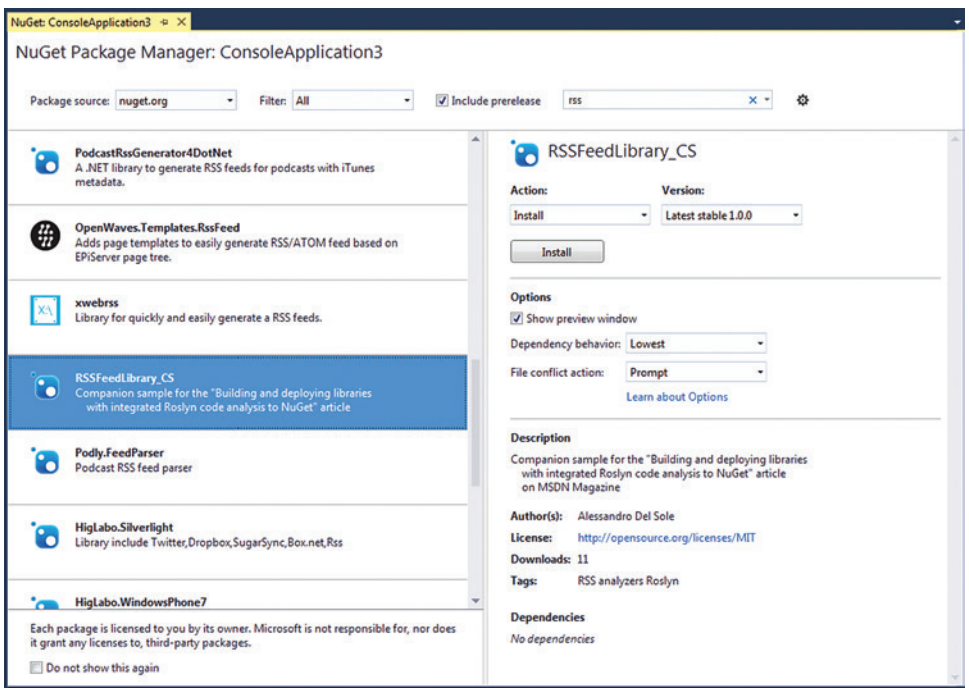


Figure 13 **The NuGet Package Is Available to the Public from the Online Repository**

Alessandro Del Sole *has been a Microsoft MVP since 2008. Awarded MVP of the Year five times, he has authored many books, eBooks, instructional videos, and articles about .NET development with Visual Studio. You can follow him on Twitter @progalex.*

Thanks to the following technical experts for reviewing this article: *Srivatsn Narayanan and Manish Vasani*

A Split-and-Merge Expression Parser in C#

Vassili Kaplan

I published a new algorithm for parsing a mathematical expression in C++ in the May and July 2015 issues of *CVu Journal* (see items 1 and 2 in "References"). It took two articles because one astute reader, Silas S. Brown, found a bug in the first algorithm implementation, so I had to make some modifications to it. Thanks to that reader, the algorithm became more mature. I've also fixed a few smaller bugs since then. Now I'm going to provide an implementation of the corrected algorithm in C#.

It's not too likely you'll ever have to write code to parse a mathematical expression, but the techniques used in the algorithm can be applied to other scenarios, as well, such as parsing non-standard strings. Using this algorithm you can also easily define new functions that do whatever you wish (for example, make a Web request to order a pizza). With small adjustments, you can also create your own C# compiler for your new custom scripting language. Moreover, you just might find the algorithm interesting in its own right.

The Edsger Dijkstra algorithm, published more than 50 years ago in 1961, is often used for parsing mathematical expressions (see item 3 in "References"). But it's good to have an alternative that, though it has the same time complexity, is, in my opinion, easier to implement and to extend.

Note that I'm going to use the "virtual constructor" idiom for the function factory implementation. This idiom was introduced in C++ by James Coplien (see item 4 in "References"). I hope you'll find its use interesting in the C# world, as well.

This article discusses:

- The split-and-merge algorithm
- Splitting an expression into a list of cells
- Merging a list of cells
- The Virtual Constructor design pattern in C#

Technologies discussed:

C#

Code download available at:

msdn.com/magazine/msdnmag1015

The Split-and-Merge Algorithm

The demo program in **Figure 1** illustrates the split-and-merge algorithm for parsing a mathematical expression.

The algorithm consists of two steps. In the first step the string containing the expression is split into a list of Cell objects, where each Cell is defined as follows:

```
class Cell
{
  internal Cell(double value, char action)
  {
    Value = value;
    Action = action;
  }

  internal double Value  { get; set; }
  internal char   Action { get; set; }
}
```

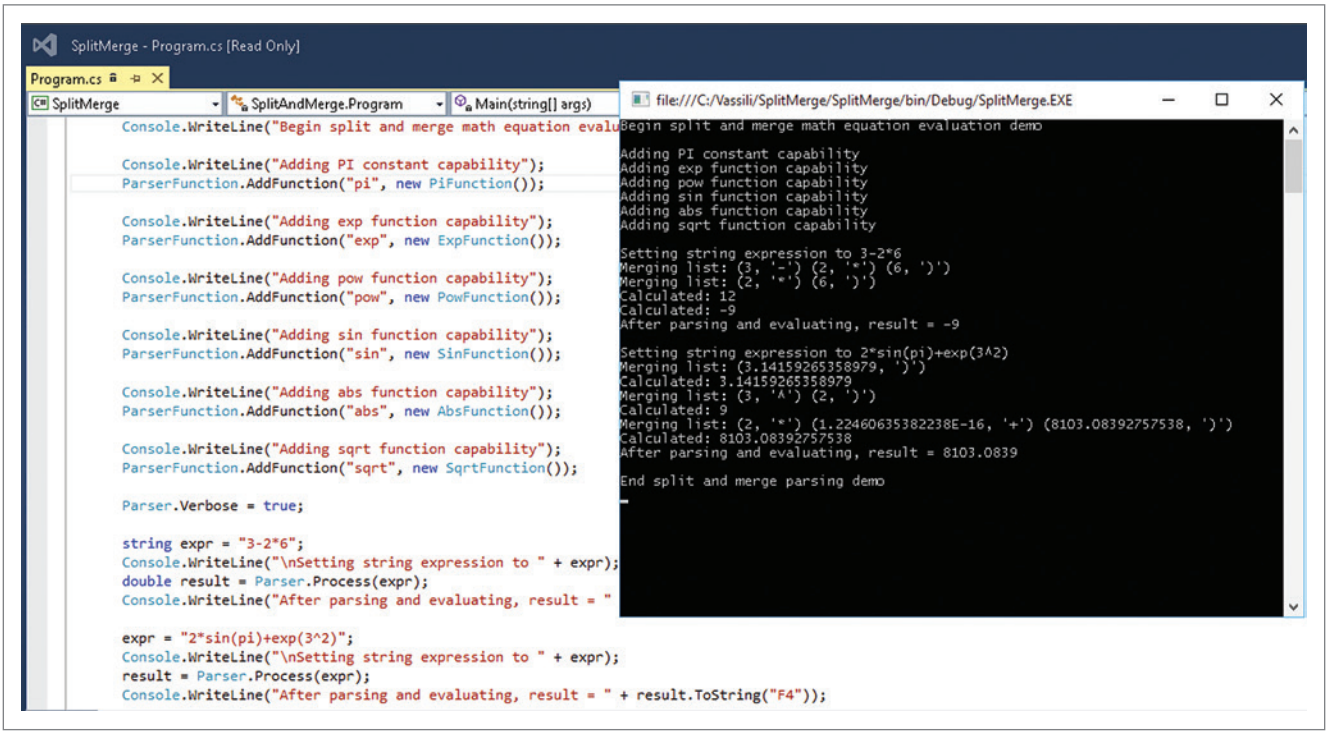


Figure 1 **A Demo Run of the Split-and-Merge Algorithm**

The action is a single character that can be any of the mathematical operators: '*' (multiplication), '/' (division), '+' (addition), '-' (subtraction) or '^' (power), or a special character denoting the end of an expression, which I hardcoded as ')'. The last element in the list of cells to be merged will always have the special action ')', that is, no action, but you can use any other symbol or a parenthesis instead.

> It's not too likely you'll ever have to write code to parse a mathematical expression, but the techniques used in the algorithm can be applied to other scenarios, as well, such as parsing non-standard strings.

In the first step, the expression is split into tokens that are then converted into cells. All tokens are separated by one of the mathematical expressions or a parenthesis. A token can be either a real number or a string with the name of a function. The ParserFunction class defined later takes care of all of the functions in the string to be parsed, or for parsing a string to a number. It may also call the whole string parsing algorithm, recursively. If there are no functions and no parentheses in the string to parse, there will be no recursion.

In the second step, all the cells are merged together. Let's see the second step first because it's a bit more straightforward than the first one.

Merging a List of Cells

The list of cells is merged one by one according to the priorities of the actions; that is, the mathematical operators. These priorities are calculated as follows:

```
static int GetPriority(char action)
{
  switch (action) {
    case '^': return 4;
    case '*':
    case '/': return 3;
    case '+':
    case '-': return 2;
  }
  return 0;
}
```

Two cells can be merged if and only if the priority of the action of the cell on the left isn't lower than the priority of the action of the cell next to it:

```
static bool CanMergeCells(Cell leftCell, Cell rightCell)
{
  return GetPriority(leftCell.Action) >= GetPriority(rightCell.Action);
}
```

Merging cells means applying the action of the left cell to the values of the left cell and the right cell. The new cell will have the same action as the right cell, as you can see in **Figure 2**.

For example, merging Cell(8, '-') and Cell(5, '+') will lead to a new Cell(8 – 5, '+') = Cell (3, '+').

But what happens if two cells can't be merged because the priority of the left cell is lower than the right cell? What happens then is a temporary move to the next, right cell, in order to try to merge it with the cell next to it, and so on, recursively. As soon as the right

cell has been merged with the cell next to it, I return to the original, left cell, and try to remerge it with the newly created right cell, as illustrated in **Figure 3**.

Note that, from the outside, this method is called with the mergeOneOnly parameter set to false, so it won't return before completing the whole merge. In contrast, when the merge method is called recursively (when the left and the right cells can't be merged because of their priorities), mergeOneOnly will be set to true because I want to return to where I was as soon as I complete an actual merge in the MergeCells method.

Also note that the value returned from the Merge method is the actual result of the expression.

Splitting an Expression into a List of Cells

The first part of the algorithm splits an expression into a list of cells. Mathematical operator precedence isn't taken into account in this step. First, the expression is split into a list of tokens. All tokens are separated by any mathematical operator or by an open or close parenthesis. The parentheses may, but don't have to, have an associated function; for example, "1- sin(1-2)" has an associated function, but "1- (1-2)" doesn't.

First, let's look at what happens when there are no functions or parentheses, just an expression containing real numbers and mathematical operators between them. In this case, I just create cells consisting of a real number and a consequent action. For example, splitting "3-2*4" leads to a list consisting of three cells:

Split ("3-2*4") = { Cell(3, '-'), Cell(2, '*'), Cell(3, ')') }

The last cell will always have a special END_ARG action, which I define as:

```
const char END_ARG = ')';
```

It can be changed to anything else, but in that case the corresponding opening parenthesis START_ARG must be taken into account, as well, which I define as:

```
const char START_ARG = '(';
```

> The list of cells is merged one by one according to the priorities of the actions; that is, the mathematical operators.

If one of the tokens is a function or an expression in parentheses, the whole split-and-merge algorithm is applied to it using recursion. For example, if the expression is "(3-1)-1," the whole algorithm in the parentheses is applied to the expression first:

Split("(3-1)-1") = Split("[SplitAndMerge("3-1")] - 1") = { Cell(2, '-'), Cell(1, ')') }

The function that performs the splitting is LoadAndCalculate, as shown in **Figure 4**.

The LoadAndCalculate method adds all of the cells to the listToMerge list and then calls the second part of the parsing algorithm, the merge function. The StringBuilder item will hold the current token, adding characters to it one by one as soon as they're read from the expression string.

The StillCollecting method checks if the characters for the current token are still being collected. This isn't the case if the current character is END_ARG or any other special "to" character (such as a comma if the parsing arguments are separated by a comma; I'll provide an example of this using the power function later). Also, the token isn't being collected anymore if the current character is a valid action or a START_ARG:

```
static bool StillCollecting(string item, char ch, char to)
{
  char stopCollecting = (to == END_ARG || to == END_LINE) ?
                        END_ARG : to;
  return (item.Length == 0 && (ch == '-' || ch == END_ARG)) ||
        !(ValidAction(ch) || ch == START_ARG || ch == stopCollecting);
}

static bool ValidAction(char ch)
{
  return ch == '*' || ch == '/' || ch == '+' || ch == '-' || ch == '^';
}
```

I know that I'm done collecting the current token as soon as I get a mathematical operator described in the ValidAction method, or parentheses defined by the START_ARG or END_ARG constants. There's also a special case involving a "-" token, which is used to denote a number starting with a negative sign.

At the end of this splitting step, all of the subexpressions in parentheses and all of the function calls are eliminated via the recursive

Figure 2 **Merge Cells Method**

```
static void MergeCells(Cell leftCell, Cell rightCell)
{
  switch (leftCell.Action)
  {
    case '^': leftCell.Value = Math.Pow(leftCell.Value, rightCell.Value);
      break;
    case '*': leftCell.Value *= rightCell.Value;
      break;
    case '/': leftCell.Value /= rightCell.Value;
      break;
    case '+': leftCell.Value += rightCell.Value;
      break;
    case '-': leftCell.Value -= rightCell.Value;
      break;
  }
  leftCell.Action = rightCell.Action;
}
```

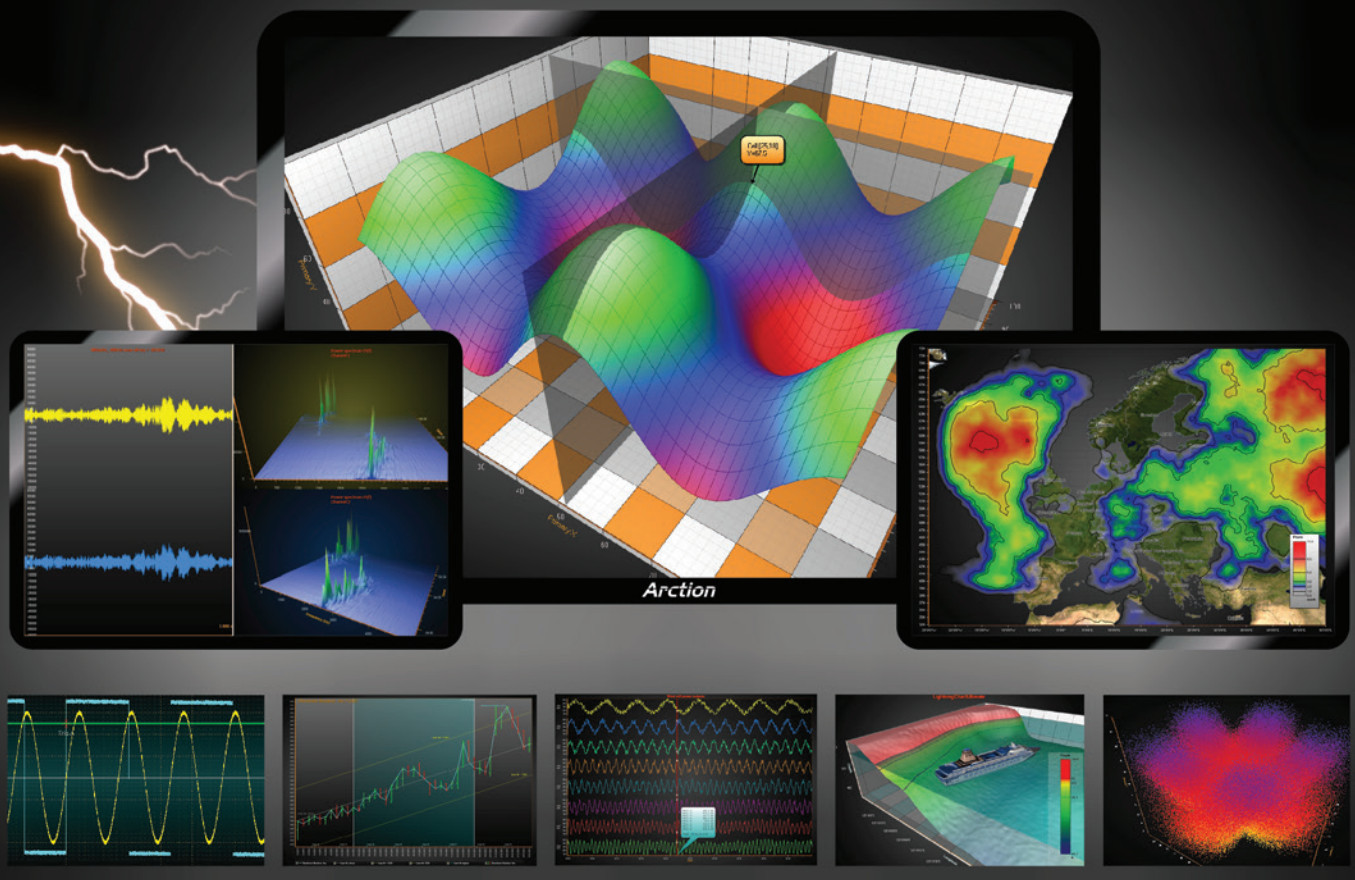
Figure 3 **Merge a List of Cells**

```
static double Merge(Cell current, ref int index, List<Cell> listToMerge,
                    bool mergeOneOnly = false)
{
  while (index < listToMerge.Count)
  {
    Cell next = listToMerge[index++];

    while (!CanMergeCells(current, next))
    {
      Merge(next, ref index, listToMerge, true /* mergeOneOnly */);
    }

    MergeCells(current, next);

    if (mergeOneOnly)
    {
      return current.Value;
    }
  }

  return current.Value;
}
```

WINDOWS 10 COMPATIBLE

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

LightningChart



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

- 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...2015
- Hundreds of examples

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.

Download a free 30-day evaluation from

www.LightningChart.com

Arction

calls to the whole algorithm evaluation. But the resulting actions of these recursive calls will always have the END_ARG action, which won't be correct in the global expression scope if the calculated expression isn't at the end of the expression to be evaluated. This is why the action needs to be updated after each recursive call, like so:

```
char action = ValidAction(ch) ? ch
                              : UpdateAction(data, ref from, ch);
```

The code for the updateAction method is in **Figure 5**.

The actual parsing of the extracted token will be in the following code of **Figure 4**:

```
ParserFunction func = new ParserFunction(data, ref from, item.ToString(), ch);
double value = func.GetValue(data, ref from);
```

If the extracted token isn't a function, this code will try to convert it to a double. Otherwise, an appropriate, previously registered function will be called, which can in turn recursively call the LoadAndCalculate method.

User-Defined and Standard Functions

I decided to implement the function factory using the virtual constructor idiom that was first published by James Coplien (see item 4 in "References"). In C#, this is often implemented using a factory method (see item 5 in "References") that uses an extra factory class to produce the needed object. But Coplien's older design pattern doesn't need an extra factory façade class and instead just constructs a new object on the fly using the implementation member m_impl that's derived from the same class:

```
private ParserFunction m_impl;
```

The special internal constructor initializes this member with the appropriate class. The actual class of the created implementation object m_impl depends on the input parameters, as shown in **Figure 6**.

> Merging cells means applying the action of the left cell to the values of the left cell and the right cell.

A dictionary is used to hold all of the parser functions. This dictionary maps the string name of the function (such as "sin") to the actual object implementing this function:

```
private static Dictionary<string, ParserFunction> m_functions =
    new Dictionary<string, ParserFunction>();
```

Users of the parser can add as many functions as they wish by calling the following method on the base ParserFunction class:

```
public static void AddFunction(string name, ParserFunction function)
{
  m_functions[name] = function;
}
```

The GetValue method is called on the created ParserFunction, but the real work is done in the implementation function, which will override the evaluate method of the base class:

```
public double GetValue(string data, ref int from)
{
  return m_impl.Evaluate(data, ref from);
}

protected virtual double Evaluate(string data, ref int from)
{
  // The real implementation will be in the derived classes.
  return 0;
}
```

The function implementation classes, deriving from the ParserFunction class, won't be using the internal constructor in **Figure 6**. Instead, they'll use the following constructor of the base class:

```
public ParserFunction()
{
  m_impl = this;
}
```

Two special "standard" functions are used in the ParserFunction constructor in **Figure 6**:

```
private static IdentityFunction s_idFunction = new IdentityFunction();
private static StrtodFunction s_strtodFunction = new StrtodFunction();
```

The first is the identity function; it will be called to parse any expression in parentheses that doesn't have an associated function:

```
class IdentityFunction : ParserFunction
{
  protected override double Evaluate(string data, ref int from)
  {
    return Parser.LoadAndCalculate(data, ref from, Parser.END_ARG);
  }
}
```

The second function is a "catchall" that's called when no function is found that corresponds to the last extracted token. This will happen

Figure 4 **LoadAndCalculate Method**

```
public static double LoadAndCalculate(string data, ref int from, char to = END_LINE)
{
  if (from >= data.Length || data[from] == to)
  {
    throw new ArgumentException("Loaded invalid data: " + data);
  }
  List<Cell> listToMerge = new List<Cell>(16);
  StringBuilder item = new StringBuilder();

  do // Main processing cycle of the first part.
  {
    char ch = data[from++];
    if (StillCollecting(item.ToString(), ch, to))
    { // The char still belongs to the previous operand.
      item.Append(ch);
      if (from < data.Length && data[from] != to)
      {
        continue;
      }
    }

    // I am done getting the next token. The getValue() call below may
    // recursively call loadAndCalculate(). This will happen if extracted
    // item is a function or if the next item is starting with a START_ARG '('
    ParserFunction func = new ParserFunction(data, ref from, item.ToString(), ch);
    double value = func.GetValue(data, ref from);

    char action = ValidAction(ch) ? ch
                                  : UpdateAction(data, ref from, ch, to);
    listToMerge.Add(new Cell(value, action));
    item.Clear();

  } while (from < data.Length && data[from] != to);

  if (from < data.Length && (data[from] == END_ARG || data[from] == to))
  { // This happens when called recursively: move one char forward.
  from++;
  }

  Cell baseCell = listToMerge[0];
  int index = 1;

  return Merge(baseCell, ref index, listToMerge);
}
```

Figure 5 **Update Action Method**

```
static char UpdateAction(string item, ref int from, char ch, char to)
{
  if (from >= item.Length || item[from] == END_ARG || item[from] == to)
  {
    return END_ARG;
  }

  int index = from;
  char res = ch;
  while (!ValidAction(res) && index < item.Length)
  { // Look to the next character in string until finding a valid action.
    res = item[index++];
  }

  from = ValidAction(res) ? index
                          : index > from ? index - 1
                                         : from;
  return res;
}
```

Figure 6 **Virtual Constructor**

```
internal ParserFunction(string data, ref int from, string item, char ch)
{
  if (item.Length == 0 && ch == Parser.START_ARG)
  {
    // There is no function, just an expression in parentheses.
    m_impl = s_idFunction;
    return;
  }

  if (m_functions.TryGetValue(item, out m_impl))
  {
    // Function exists and is registered (e.g. pi, exp, etc.).
    return;
  }

  // Function not found, will try to parse this as a number.
  s_strtodFunction.Item = item;
  m_impl = s_strtodFunction;
}
```

when the extracted token is neither a real number nor an implemented function. In the latter case, an exception will be thrown:

```
class StrtodFunction : ParserFunction
{
  protected override double Evaluate(string data, ref int from)
  {
    double num;
    if (!Double.TryParse(Item, out num)) {
      throw new ArgumentException("Could not parse token [" + Item + "]");
    }
    return num;
  }
  public string Item { private get; set; }
}
```

All other functions can be implemented by the user; the simplest is a pi function:

```
class PiFunction : ParserFunction
{
  protected override double Evaluate(string data, ref int from)
  {
    return 3.141592653589793;
  }
}
```

A more typical implementation is an exp function:

```
class ExpFunction : ParserFunction
{
  protected override double Evaluate(string data, ref int from)
  {
    double arg = Parser.LoadAndCalculate(data, ref from, Parser.END_ARG);
    return Math.Exp(arg);
  }
}
```

Earlier I said I'd provide an example using a power function, which requires two arguments separated by a comma. This is how to write a function requiring multiple arguments separated by a custom separator:

```
class PowFunction : ParserFunction
{
  protected override double Evaluate(string data, ref int from)
  {
    double arg1 = Parser.LoadAndCalculate(data, ref from, ',');
    double arg2 = Parser.LoadAndCalculate(data, ref from, Parser.END_ARG);

    return Math.Pow(arg1, arg2);
  }
}
```

Any number of functions can be added to the algorithm from the user code, like so:

```
ParserFunction.AddFunction("pi",  new PiFunction());
ParserFunction.AddFunction("exp", new ExpFunction());
ParserFunction.AddFunction("pow", new PowFunction());
```

A dictionary is used to hold all of the parser functions.

Wrapping Up

The split-and-merge algorithm presented here has O(n) complexity if n is the number of characters in the expression string. This is so because each token will be read only once during the splitting step and then, in the worst case, there will be at most 2(m - 1) – 1 comparisons in the merging step, where m is the number of cells created in the first step.

So the algorithm has the same time complexity as the Dijkstra algorithm (see item 3 in "References"). It might have a slight disadvantage compared to the Dijkstra algorithm because it uses recursion. On the other hand, I believe the split-and-merge algorithm is easier to implement, precisely because of the recursion, and easier also to extend with the custom syntax, functions, and operators. ■

References

1. V. Kaplan, "Split and Merge Algorithm for Parsing Mathematical Expressions," *CVu*, 27-2, May 2015, bit.ly/1Jb470I

2. V. Kaplan, "Split and Merge Revisited," *CVu*, 27-3, July 2015, bit.ly/1UYHmE9

3. E. Dijkstra, Shunting-yard algorithm, bit.ly/1fEwLI

4. J. Coplien, "Advanced C++ Programming Styles and Idioms" (p. 140), Addison-Wesley, 1992

5. E. Gamma, R. Helm, R. Johnson and J. Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software," Addison-Wesley Professional Computing Series, 1995

Vassili Kaplan *is a former Microsoft Lync developer. He is passionate about programming in C# and C++. He currently lives in Zurich, Switzerland, and works as a freelancer for various banks. You can reach him at iLanguage.ch.*

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

Develop a Windows 10 App with the Microsoft Band SDK

Kevin Ashley

Microsoft Bing predicted wearables would be the hottest trend in technology in 2015 (binged.it/1hCBv9A), ahead of personal digital assistants, home automation, 3D printing and virtual reality gaming.

One such device, Microsoft Band, combines most sensors people need to track their health, fitness, sleep quality and more. It packs more sensors than many of the more expensive devices available in the market.

As a mobile app developer, I'm interested in creating useful applications with Microsoft Band. Some of the examples used in this code I used in my Active Fitness app (activefitness.co), which has more than 2 million users on Windows, iOS and Android. Most consumer apps need to work on multiple platforms such as Windows, iOS and Android to reach a maximum audience. In this article, I'll explain how to develop Windows 10 apps for Band and explore cross-platform options. It was an honor to participate in the early versions of the Band SDK. Now that it has been made publically available, I can share code examples, enhanced with the latest releases of the Band SDK and the developer community.

This article discusses:

- Developing a Windows 10 app with the Microsoft Band SDK
- Exploring cross-platform options with Xamarin

Technologies discussed:

Windows 10, Microsoft Band SDK, Visual Studio 2015, iOS, Android

Code download available at:

bit.ly/1MIKIIK

Gravity Hero: Windows 10 Sample App

I wanted to build something fun that showcases Band features and at the same time works almost like a game. So I built a Universal Windows app called "Gravity Hero," as shown in **Figure 1**. The idea is simple: You jump wearing the Band and the device tells you whether you made your best jump ever. And because I built a Universal Windows app, it'll work on any Windows 10 device. I also

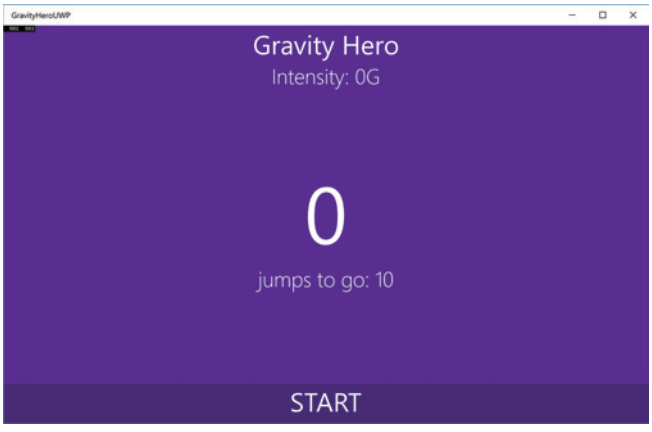


Figure 1 **Gravity Hero Sample App Running on Windows 10**

added sample code on GitHub for a native Android app to demonstrate how you can easily target other devices. In addition, the Band Developer Web site includes links to documentation and SDK code samples (developer.microsoftband.com).

The Microsoft Band team provides SDKs for Windows, iOS and Android. In addition, there's a cross-platform SDK solution with Xamarin, technology that enables a great code reuse. Which technology should you choose? With support of Microsoft and the community, you have the following options when developing for the Band device:

- **Microsoft Band SDK (Microsoft)** for native development with Windows, iOS and Android (bit.ly/1JfiFvW)
- **Web Tile SDK (Microsoft)** for quickly delivering information to the Band from any Web source in just a few easy steps (bit.ly/1h94CjZ)
- **Cloud API (Microsoft)** for accessing RESTful APIs with comprehensive fitness and health data in an easy-to-consume JSON format (bit.ly/1MIBOL7)
- **Cross-Platform SDK (Xamarin)** for use in Xamarin cross-platform apps, targeting iOS, Android and Windows; with Xamarin you can use a single code base for all platforms (bit.ly/1EfhqjK)

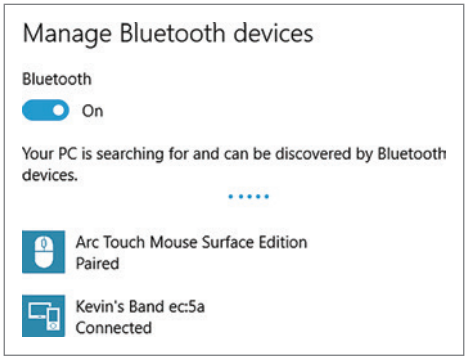


Figure 2 **Connecting Microsoft Band to a Windows 10 PC**

Connecting the Band to a Windows 10 PC

Before you start with development, you'll need to connect the Band to a Windows 10 PC. Windows 10 includes Bluetooth support, so this is very easy to do:

On your Band, flip to the Settings tile, click the Bluetooth icon and switch to Pairing. Your Band is now in pairing mode. On your Windows 10 PC, go to Settings or type Bluetooth in Cortana and open the Bluetooth settings page, as shown in **Figure 2**. Note the status of your Band in the list of Bluetooth devices. Band names usually start with your name and a code, unless you changed it to something else (in this case, I have "Kevin's Band ec:5a"). If the status says "Connected," you're good to go; otherwise, tap Pair and follow the prompts.

Creating a Windows 10 App in Visual Studio

You can start with the sample app I created on GitHub (bit.ly/1U2sLup). Remember that if you decide to create your app from scratch, you need to include Bluetooth device capability in your manifest; Visual Studio by default doesn't include this capability in the application template:

```
<Capabilities>
  <DeviceCapability Name="bluetooth" />
</Capabilities>
```

If you add the Band to your own app or create a Windows 10 app from scratch you'll also need to get a Microsoft Band NuGet package. There are several packages out there, some created by Microsoft and some added by the community (the Xamarin.Microsoft.Band package, for example). For the purpose of the Universal Windows app, I added the Microsoft package, as shown in **Figure 3**.

The Microsoft Band SDK allows developers to access sensors on the Band, creating and updating tiles and personalizing the device. Your app can send notifications, including haptics, to the Band. The Band itself has a nice, clean and simple look to it, as shown in **Figure 4**.

The Band is packed with sensors (the full list can be seen in **Figure 5**); for the purpose of my Gravity Hero sample app, I focused on the Accelerometer sensor.

Designing a Data Model

As always, I start with a data model in Visual Studio, as shown in **Figure 6**. A good data model always helps, as it separates data into a separate tier, which can be easily shared across your apps and

If you add the Band to your own app or create a Windows 10 app from scratch you'll also need to get a Microsoft Band NuGet package.

If you already have a Windows, iOS or Android app, simply use a native Band SDK for each of these platforms. If you're starting from scratch or want to target all platforms with the same code, you might want to take a look at Xamarin. The choice is yours. The platform-friendly approach gives you all the choices you need to start building apps. In this article, I'll look at both options.

Figure 3 **Microsoft Band NuGet Packages**

Figure 4 **Microsoft Band**

Figure 5 **Sensors Available on the Microsoft Band**

Sensor	Details
Accelerometer	Provides X, Y and Z acceleration in g units. 1 g = 9.81 meters per second squared (m/s2).
Gyroscope	Provides X, Y and Z angular velocity in degrees per second (°/sec) units.
Distance	Provides the total distance in centimeters, current speed in centimeters per second (cm/s), current pace in milliseconds per meter (ms/m), and the current pedometer mode (such as walking or running).
Heart Rate	Provides the number of beats per minute; also indicates if the heart rate sensor is fully locked onto the wearer's heart rate. The data returned should be used only in resting mode. The Band SDK doesn't provide access to the heart rate values optimized for any other activity.
Pedometer	Provides the total number of steps the wearer has taken.
Skin Temperature	Provides the current skin temperature of the wearer in degrees Celsius.
UV	Provides the current ultraviolet radiation exposure intensity.
Band Contact	Provides the current state of the Band as being worn/not worn.
Calories	Provides the total number of calories the wearer has burned.

even a cloud back end if you decide to have a Web site displaying your data. I want to make my data model flexible enough to collect data from sensors and present them in the Gravity Hero app. I can reuse my data model in other apps, so it's worth placing it in a separate folder in my Visual Studio solution.

depending on the serializer. Another advantage to the SensorReading class is that it provides a normalized form because the class holds a 3D vector reading or Vector3. As shown in **Figure 7**, I also provide a Value member that returns a normalized vector (a very useful thing when you deal with 3D calculations).

A good data model always helps, as it separates data into a separate tier, which can be easily shared across your apps and even a cloud back end if you decide to have a Web site displaying your data.

The data model for the Gravity Hero app is using a helper class conveniently called ViewModel. I can use a myriad of existing Model-View-ViewModel (MVVM) helper libraries, but in this example I simply implemented one to make the code as transparent as possible. The ViewModel class implements the INotifyPropertyChanged interface. This is a fairly standard approach in .NET apps when you want to reflect changes in your data model in the UI. In the implementation of my sample app, I'd like to make sure that the class notifies the UI via the PropertyChanged mechanism.

SensorReading is the class to catch Band sensor readings and notify anything in the UI that's subscribed to the data model changes. Let's look at this class more closely. The class derives from the ViewModel introduced earlier and also includes attributes that might be useful to serialize data and send it across the wire (to your cloud storage, for example). DataContract attribute on the SensorReading class does just that, and DataMember attributes on individual data members allow for data to be serialized, either into JSON or XML,

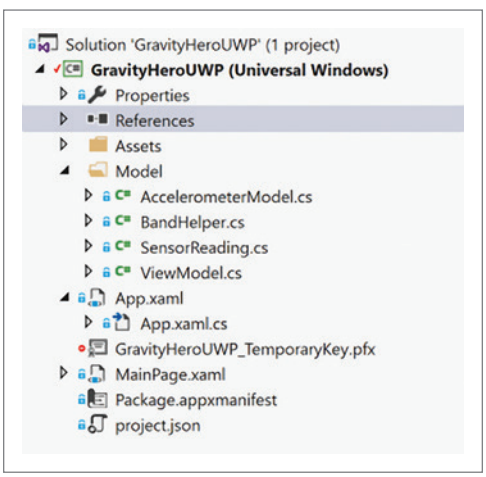


Figure 6 **Visual Studio Solution with the Sample Project and Data Model**

BandModel is used to manage the Band, as shown in **Figure 8**. This is the manager class that helps in case I have multiple Band devices connected to my PC; it also can tell me if any Band is connected.

AccelerometerModel is designed specifically for the Gravity Hero game. Gravity is the force that can effectively be measured by the built-in Accelerometer sensor in the Band. Now you see how the data model classes are created; you can add additional classes for any of the Band sensors you want to use in your apps. When I need to initialize the Accelerometer class in Init method, I subscribe to several events conveniently presented by the Band SDK:

```
if (BandModel.IsConnected)
{
  BandModel.BandClient.SensorManager.
    Accelerometer.ReadingChanged +=
    Accelerometer_ReadingChanged;
  BandModel.BandClient.SensorManager.
    Accelerometer.ReportingInterval =
    TimeSpan.FromMilliseconds(16.0);
  BandModel.BandClient.SensorManager.
    Accelerometer.StartReadingsAsync(
    new CancellationToken());
  totalTime = 0.0;
}
```

The first event is ReadingChanged. This is the event that gives me data from the accelerometer sensor based on the ReportingInterval time period that I

Modern Apps LIVE!

MOBILE, CROSS-DEVICE & CLOUD DEVELOPMENT

Presented in Partnership with Magenic

ORLANDO
ROYAL PACIFIC RESORT AT
UNIVERSAL ORLANDO

NOV 16-20

LIVE! 360
TECH EVENTS WITH PERSPECTIVE

5 Great Conferences 1 Great Price

Leading the Modern Apps Way

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

What sets Modern Apps Live! apart is the singular topic focus; sessions build on each other as the conference progresses, leaving you with a holistic understanding of modern applications, which means a complete picture of what goes into building a modern app for Windows 10, iPad, and Windows Phone devices that all interact with state-of-the-art backend services running in public or private clouds.

SQL Server LIVE!
SQL SERVER FOR MODERN DEVELOPERS

Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS

SharePoint LIVE!
TRAINING FOR COLLABORATION

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

TECHMENTOR
IN-DEPTH TRAINING FOR IT PROS

REGISTER BY OCTOBER 14 AND **SAVE $300!**

Use promo code MALOCT1

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

SUPPORTED BY
Visual Studio msdn magazine Visual Studio MAGAZINE

PRODUCED BY
1105 MEDIA YOUR GROWTH. OUR BUSINESS.

MODERNAPPSLIVE.COM

Figure 7 **SensorReading Class for the Sensor Data Model**

```
[DataContract]
public class SensorReading : ViewModel
{
  DateTimeOffset _timestamp;

  [DataMember]
  public DateTimeOffset Timestamp
  {
    get { return _timestamp; }
    set
    {
      SetValue(ref _timestamp, value, "Timestamp");
    }
  }

  double _x;

  [DataMember]
  public double X
  {
    get { return _x; }
    set
    {
      SetValue(ref _x, value, "X", "Value");
    }
  }

  double _y;

  [DataMember]
  public double Y
  {
    get { return _y; }
    set
    {
      SetValue(ref _y, value, "Y", "Value");
    }
  }

  double _z;

  [DataMember]
  public double Z
  {
    get { return _z; }
    set
    {
      SetValue(ref _z, value, "Z", "Value");
    }
  }


  public double Value
  {
    get
    {
      return Math.Sqrt(X * X + Y * Y + Z * Z);
    }
  }
}
```

define. For reading Accelerometer values, I use a 16 ms threshold. It's important to keep the reporting interval as small as possible for precision, but at the same time consider that battery consumption increases with extensive use of the sensor. Next, I call StartReadingsAsync, the method that starts reading values from the sensor and sends it back to the app. This method just starts a listener for sensor data readings. The data is passed to the ReadingChanged event.

In the ReadingChanged event I capture the reading and recalculate values in my data model:

```
void Accelerometer_ReadingChanged(object sender,
  BandSensorReadingEventArgs<IBandAccelerometerReading> e)
{
  SensorReading reading = new SensorReading {
    X = e.SensorReading.AccelerationX, Y = e.SensorReading.AccelerationY,
    Z = e.SensorReading.AccelerationZ };
  _prev = _last;
  _last = reading;
  Recalculate();
}
```

The Recalculate model method is where most of my logic happens (see **Figure 9**). I want to fire Changed event back to the app when

Figure 8 **BandModel for Managing the Bands**

```
public class BandModel : ViewModel
{
  static IBandInfo _selectedBand;

  public static IBandInfo SelectedBand
  {
    get { return BandModel._selectedBand; }
    set { BandModel._selectedBand = value; }
  }

  private static IBandClient _bandClient;
  public static IBandClient BandClient
  {
    get { return _bandClient; }
    set
    {
      _bandClient = value;
    }
  }


  public static bool IsConnected
  {
    get {
      return BandClient != null;
    }

  }

  public static async Task FindDevicesAsync()
  {
    var bands = await BandClientManager.Instance.GetBandsAsync();
    if (bands != null && bands.Length > 0)
    {
      SelectedBand = bands[0]; // Take the first band

    }
  }

  public static async Task InitAsync()
  {
    try
    {
      if (IsConnected)
        return;

      await FindDevicesAsync();
      if (SelectedBand != null)
      {
        BandClient =
          await BandClientManager.Instance.ConnectAsync(SelectedBand);
        // Connected!
          BandModel.BandClient.NotificationManager.VibrateAsync(
            Microsoft.Band.Notifications.VibrationType.ExerciseRunLap);
      }
    }
    catch (Exception x)
    {
      Debug.WriteLine(x.Message);
    }
  }
}
```

I exceed values achieved earlier when I started the game. Remember, I implement a Gravity Hero game here, so I'm looking for the best results. I want to make sure that I use Dispatcher class, because Band sensor events may be triggered on non-UI threads, so I need to marshal the code to the UI thread for my Changed event.

Figure 9 **The Recalculate Model Method**

```
DateTimeOffset _startedTime = DateTimeOffset.MinValue;
double totalTime = 0.0;
double lastTime = 0.0;
SensorReading _prev;
SensorReading _last;
double MIN = 0.4;
void Recalculate()
{
  if (_last.Value <= MIN)
  {
    if (_startedTime > DateTimeOffset.MinValue)
      lastTime = (DateTimeOffset.Now - _startedTime).TotalSeconds;
    else
      _startedTime = DateTimeOffset.Now;
    }
    else
    {
      if (_startedTime > DateTimeOffset.MinValue)
      {
        lastTime = (DateTimeOffset.Now - _startedTime).TotalSeconds;
        totalTime += lastTime;
        lastTime = 0.0;
        _startedTime = DateTimeOffset.MinValue;
          CoreApplication.MainView.CoreWindow.Dispatcher.RunAsync(
          CoreDispatcherPriority.Normal,        () =>
        {
          if (Changed != null)
            Changed(_last.Value);
            });

        }
      }
    }
i   }
}
```

Figure 10 **UI Updated in Changed Event**

```
void _accelerometerModel_Changed(double force)
{

  bandCount++;
  UpdateCount();
  if (force > maxForce)
  {
    maxForce = force;
    heroText.Text = String.Format("Intensity {0:F2}G", maxForce);
  }
  if (!isAchievementUnlocked && bandCount >= maxCount*0.2)
  {
    Speak("Just a few more!");
    isAchievementUnlocked = true;
  }
  if (!isSecondAchievementUnlocked && isAchievementUnlocked &&
    bandCount >= maxCount * 0.8)
  {
    Speak("Almost there!");
    isAchievementUnlocked = true;
  }
    BandModel.BandClient.NotificationManager.VibrateAsync(
      Microsoft.Band.Notifications.VibrationType.ExerciseRunLap);
  // Speak(bandCount.ToString()+"!");
}
```

My UI needs to be updated when events come back from the Band (see **Figure 10**). This is done in the Changed event, which I subscribed to in my MainPage.xaml. I display G-force and count achievements. I also invoke the Band haptic feedback via the VibrateAsync method. As a Band user for many months, I really like the ability to send haptic notifications (just remember not to abuse it and notify users only when appropriate).

Microsoft Band provides powerful SDKs and community support for multiple platforms.

Let the Fun Begin

Now, with everything in place, let the fun begin. Build and launch the app: Remember, the app runs on any Windows 10 device. Because I'm using the Band sensors, I don't care about PC or phone support for Accelerometers. I'll be using a PC to display information from the Band, while the Band will be doing all the hard work.

As shown in **Figure 11**, when you launch the app for the first time whether from Visual Studio or from the tile, Windows 10 automatically prompts you to confirm that you want to grant Gravity Hero app access to the Band; just select Yes.

When Gravity Hero connects to the Band, you'll feel the Band vibrate. This is done intentionally, to let you know that everything is set up for jumps. I also added audible prompts to notify you that you can start the action:

```
BandModel.BandClient.NotificationManager.VibrateAsync(
  Microsoft.Band.Notifications.VibrationType.ExerciseRunLap);
```

Now, keep jumping. The app will count your jumps and congratulate you on each achievement.

All sample code for this article is available on GitHub: bit.ly/1MIKIIK. To use this source you can use Visual Studio 2015 and Windows 10. The project uses the Microsoft Band SDK NuGet package.

Wrapping Up

Microsoft Band provides powerful SDKs and community support for multiple platforms—Windows, Android and iOS. Developers can use Microsoft SDKs and community components from Xamarin, GitHub and the developer community to extend apps to use Microsoft Band. You can use the code in this article to integrate Microsoft Band into your own apps for Windows 10. ■

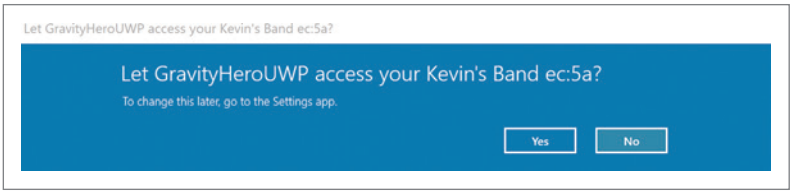


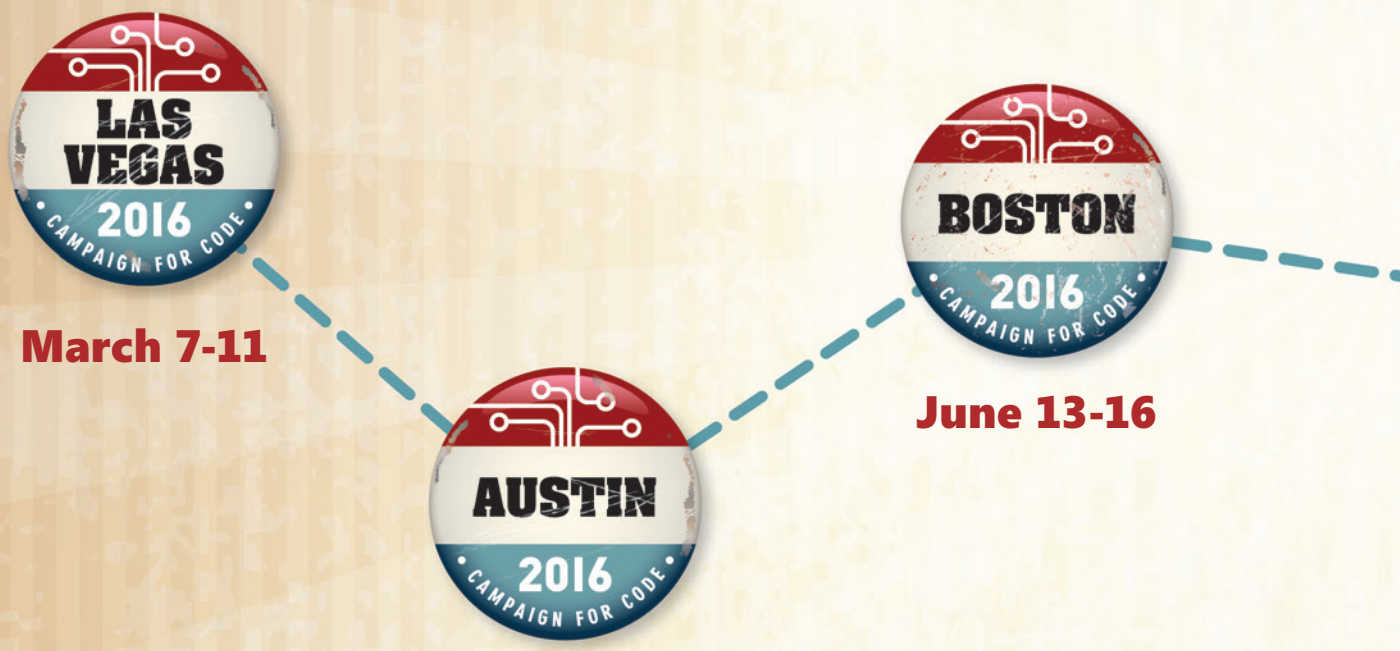
Figure 11 **Windows 10 Dialog Automatically Requesting Access to the Band**

Kevin Ashley *is a senior game developer evangelist for Microsoft. He's a co-author of "Professional Windows 8 Programming" (Wrox, 2012) and a developer of top apps and games, most notably Active Fitness, which has more than 2 million users (activefitness.co). He often presents on technology at various events, industry shows and Web casts. In his role, he works with startups and partners, advising on software design, business and technology strategy, architecture, and development. You can follow Ashley on his blog at kevinashley.com and on Twitter @kashleytwit.*

Thanks to the following Microsoft technical expert for reviewing this article: *Jaime Rodriguez*

Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS

JOIN US



SUPPORTED BY

Microsoft | Visual Studio | msdn magazine | Visual Studio MAGAZINE

PRODUCED BY

1105 MEDIA
YOUR GROWTH. OUR BUSINESS.

ON THE 2016 CAMPAIGN FOR CODE TRAIL!

vslive.com

JAMES MCCAFFREY

Linear Discriminate Analysis Using C#

The goal of a binary classification problem is to predict the value of a variable that can take on one of two possible values. For example, you might want to predict the change in the stock price of some company (increase or decrease) based on predictor variables like average number of shares sold, number of insider transactions, price to earnings ratio and so forth. Or you might want to predict the political inclination of a person (liberal or conservative) based on age, income, education level and so on.

There are roughly a dozen major algorithms that can be used for binary classification. Linear discriminate analysis (LDA) is one of the oldest approaches, dating from the 1930s. This article explains what LDA is, describes how it works and shows you how to implement LDA with code.

Realistically, it's unlikely you'll ever need to write LDA code. Nevertheless, there are three reasons you might find this article useful. First, understanding how to program LDA gives you a thorough grasp of how LDA works in case you ever come across it. Second, some of the coding techniques used when implementing LDA can be useful in other, more common programming scenarios. Finally, the ideas behind LDA are extraordinarily clever and you might find LDA interesting for its own sake.

The best way to get a feel for what LDA binary classification is and to see where this article is headed is to take a look at the demo program in **Figure 1**.

The goal of the demo program is to predict the political inclination, liberal or conservative, of a person based on the person's age and annual income. The demo uses a tiny training data set with just eight items to create the LDA prediction model. Both age and income have been normalized in some way so that their magnitudes are both roughly the same.

Political inclination has been encoded so that liberal is 0 and conservative is 1. Unlike many binary classification algorithms, LDA can use any type of encoding for the variable to predict. So political inclination could have been encoded as -1 and +1, or "A" and "B."

Basic LDA binary classification can work with any number of predictor variables. And it's possible to extend basic LDA to predict a variable that can take one of three or more values. For example, you could predict political inclination where the possible values are liberal, conservative and moderate. This article describes only binary classification.

Code download available at msdn.com/magazine/msdnmag1015.

The key to LDA is something called the linear discriminate, usually represented by lowercase "w." Using the eight data items, the demo calculates w = (-0.868, -0.496). The w array will have the same number of values as there are predictor variables. When computing w, behind the scenes the demo calculates the means for each of the two classes, then uses the means to calculate scatter matrices for each class, and finally uses the scatter matrices to calculate a combined within-class scatter matrix. The within-class matrix is needed to calculate w.

The demo uses a tiny training data set with just eight items to create the LDA prediction model.

After calculating w, the demo uses it to predict the class for a new person who has normalized age = 4.0 and normalized income = 5.0. The LDA prediction is that the person has a liberal political inclination.

This article assumes you have at least intermediate programming skills, but doesn't assume you know anything about the LDA classification algorithm. The demo is coded using C#, but you shouldn't have much trouble if you want to refactor the code to another language such as Visual Basic .NET or JavaScript.

Understanding LDA Binary Classification

The main ideas of LDA binary classification are illustrated in the two graphs shown in **Figure 2**. The top graph shows the data from the demo program. The three blue dots represent the three people who are liberal. The five red dots are the conservatives. The yellow dot at (4.0, 5.0) represents a person with unknown political inclination. Even for such a simple problem, it's not obvious if the unknown person should be classified as a liberal or a conservative.

Note that the only reason the demo data could be graphed as shown in **Figure 2** is that there are just two predictor variables. If there were more than two predictor variables, it wouldn't be possible to see the data so nicely in a 2D graph, but the same principles of LDA would apply. No matter how many predictor variables there are in binary LDA, there will only be two classes. It's easy to confuse the number of predictor variables (two or more)

with the number of values that the variable to predict can take (always exactly two for binary classification).

Most binary classification algorithms would try to find a line (technically a vector) that explicitly separates the two classes. For example, in the top graph in **Figure 2**, you can imagine a hypothetical separating line that runs from about (3, 9) down to (5, 0). Any unknown data point on the left side of the line would be classified as liberal, and any point on the right side would be a conservative. LDA works in an entirely different way.

The first step in LDA is to find a line called the discriminate. In **Figure 2**, the discriminate line is colored green, and the discriminate is identified as (0.868, 0.496). In LDA, the discriminate line is always identified by a single end point and the starting point is always implicitly (0, 0, . ., 0).

But wait a minute; the output of the demo program in **Figure 1** shows the discriminate is (-0.868, -0.496) rather than (+0.868, +0.496). As it turns out, you can multiply the components of the discriminate by any constant and there will be no effect on the result. Therefore, to make the bottom graph in **Figure 2** look nicer and to illustrate this important idea, I used (+0.868, +0.496) for w rather than the actual calculated values, which were negative. In other words, I multiplied both components by -1.

Put another way, the discriminate could be identified by any point on the green line in **Figure 2**, such as -2 * (-0.868, -0.496) = (1.736, 0.992). In LDA, the standard, but by no means universal, approach is to scale the discriminate so that it has length 1 from the origin. Notice that the length from (0, 0) to (0.868, 0.496) = sqrt((0 - 0.868)^2 + (0 - 0.496)^2) = sqrt(0.754 + 0.246) = sqrt(1.000) = 1.

> In LDA, the standard, but by no means universal approach, is to scale the discriminate so that it has length 1 from the origin.

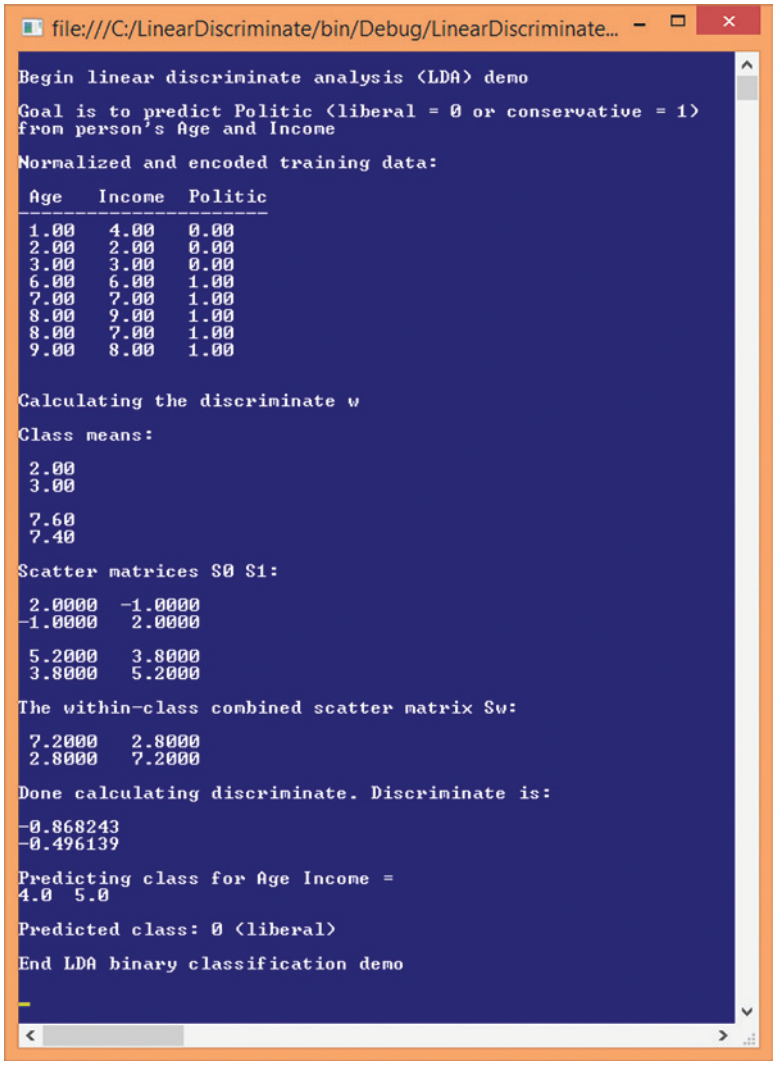


Figure 1 **Linear Descriminate Analysis Binary Classification Demo**

But what's the significance of the discriminate vector? In **Figure 2**, there are black dashed lines from each data point projected onto the discriminate line, where the dashed lines are perpendicular to the discriminate. As it turns out, the discriminate is the one line, starting at (0, 0) that simultaneously minimizes the distance of the projected points for each class, and maximizes the distance between the two groups of projected points. This idea is not at all obvious.

OK, but even if it's possible to calculate the discriminate vector for a set of data, it's still not clear how the discriminate can be used to make a prediction. In **Figure 2**, the purple diamond labeled "mean" is the average data point of the two class means. You can see the mean is halfway between the two groups of data points. Now imagine a perpendicular dashed line from the purple mean down to the green discriminate line. The projection line would hit at about (5.2, 3.0).

And now imagine a perpendicular line from the yellow point to classify down to the green discriminate line. Because the projection of the point to classify falls to the left of the projection from the mean, it's closer to the projections of the liberal data points, and therefore is classified as liberal. If the projection from the unknown point to the discriminate line had fallen on the other side of the projection from the mean, the point would have been classified as conservative. An incredibly clever idea.

Implementing LDA Binary Classification

The overall structure of the demo program, with a few WriteLine statements removed and minor edits to save space, is presented in **Figure 3**. To create the demo program, I launched Visual Studio and created a new C# console application named LinearDiscriminate. The demo has no significant .NET version dependencies

so any version of Visual Studio should work. After the template code loaded into the editor, I deleted all using statements except for the single reference to the top-level System namespace. In the Solution Explorer window, I renamed file Program.cs to LinearDiscriminateProgram.cs and allowed Visual Studio to automatically rename class Program for me.

The demo program is too long to present in its entirety here, but you can find the complete source code in the download that accompanies this article. I used a static method approach rather than an object-oriented programming approach.

The Main method begins by setting up the hardcoded eight-item training data set into an array-of-arrays-style matrix:

```
double[][] data = new double[8][];
data[0] = new double[] { 1.0, 4.0, 0 };
data[1] = new double[] { 2.0, 2.0, 0 };
// Etc. for [2] through [6]
data[7] = new double[] { 9.0, 8.0, 1 };
ShowMatrix(data, 2, true);
```

In a non-demo scenario, you'd probably read data from a text file into memory using a method named something like LoadData. Calculating the discriminate is performed like so:

```
double[][] w = Discriminate(data, true);
Console.WriteLine("Discriminate is: ");
ShowMatrix(w, 6, true);
```

Notice that the return value of the Discriminate method is an array-of-arrays matrix rather than an array. Most of the operations used in LDA are matrix operations, such as matrix multiplication and matrix inversion. Here, instead of an array with two cells, w is a matrix with two rows and one column. Such one-column matrices are sometimes called column vectors. Unless you work with matrices often, it can take a while to get accustomed to working with them instead of arrays.

Figure 2 **LDA Prediction Using the Discriminate Vector**

In **Figure 1**, you can see that several intermediate objects are calculated and displayed. The display statements are inside method Discriminate and its helper methods and are intended to help you understand LDA. You'd probably remove the display statements in production code.

The statements setting up the item to predict are:

```
double[] x = new double[] { 4.0, 5.0 };
Console.WriteLine("Predicting class for Age Income =");
ShowVector(x, 1);
```

Here, for calling convenience, the item to predict is housed in an ordinary numeric array, even though it will have to be converted to a one-column matrix later. The prediction statements are:

```
int c = Prediction(data, x, w);
Console.Write("Predicted class: " + c);
if (c == 0)
  Console.WriteLine(" (liberal)");
else
  Console.WriteLine(" (conservative)");
```

The prediction method accepts the data matrix in order to compute the mean-of-means as shown in **Figure 2**. The method also requires the item to predict, x, and the discriminate vector, w.

Calculating the Discriminate

Method Discriminate calculates the LDA discriminate vector. The code, with WriteLine and display statements removed, is surprisingly short because most of the work is done by helper methods:

```
static double[][] Discriminate(double[][] data, bool unitize)
{
  double[][] mean0 = Mean(data, 0);
  double[][] mean1 = Mean(data, 1);
  double[][] Sw = ScatterWithin(data);
  double[][] SwInv = MatrixInverse(Sw);
  double[][] diff = MatrixSubtract(mean0, mean1);
  double[][] w = MatrixProduct(SwInv, diff);
  if (unitize == true) return Unitize(w);
  else return w;
}
```

The math behind the LDA discriminate is fairly complex, but the results are rather simple. Helper method Mean computes the mean of a given class (0 or 1). For example, the three data items that represent liberal (class 0) are (1, 4), (2, 2), and (3, 3). Their mean is ((1+2+3) / 3, (4+2+3)/3) = (2.0, 3.0).

The scatter-within matrix is a measure of how variable the data set is. With the two class means, mean0 and mean1, and the scatter-within matrix Sw in hand, the discriminate is the matrix product of the inverse of Sw and the matrix difference of mean0 and mean1.

You can find several resources on the Internet that explain the rather complex math that derives the equation for the discriminate. Be aware that there are many different versions of the derivation for w. In particular, you may see references to covariance and to scatter-between (Sb). Covariance is a mathematical entity that's equivalent to scatter-within. Scatter-between is a mathematical entity that's used in the derivation of the equation for the LDA discriminate, but scatter-between is not explicitly used to calculate the discriminate or to make a prediction.

Method Discriminate has a Boolean parameter named unitize that indicates whether or not to scale the discriminate to unit length, that is, a length equal to 1.0. In most situations you want to pass true as the corresponding argument.

Figure 3 **Overall Demo Program Structure**

```
using System;
namespace LinearDiscriminate
{
  class LinearDiscriminateProgram
  {
    static void Main(string[] args)
    {
      Console.WriteLine("Begin LDA demo");
      // All program control statements here
      Console.WriteLine("End LDA demo");
      Console.ReadLine();
    }

    static int Prediction(double[][] data,
      double[] x, double[][] w) { . . }

    static int Prediction(double[][] data,
      double[][] x, double[][] w) { . . }

    static double[][] Mean(double[][] data,
      int c) { . . }

    static double[][] Discriminate(double[][] data,
      bool unitize) { . . }

    static double[][] ScatterWithin(double[][] data) { . . }

    static double[][] Scatter(double[][] data, int c) { . . }

    static double[][] Unitize(double[][] vector) { . . }

    // Matrix helper methods here
  }
} // ns
```

Making a Prediction

The demo program has two overloaded Prediction methods. The first accepts the item to predict, x, as a normal numeric array:

```
static int Prediction(double[][] data, double[] x, double[][] w)
{
  double[][] xm = MatrixFromVector(x);
  return Prediction(data, xm, w);
}
```

This version of Prediction is for calling convenience and is just a wrapper around the version of Prediction that does the work:

```
static int Prediction(double[][] data, double[][] x, double[][] w)
{
  int dim = data[0].Length - 1; // at least 2
  double[][] wT = MatrixTranspose(w); // 1 x d
  double[][] m0 = Mean(data, 0); // d x 1
  double[][] m1 = Mean(data, 1); // d x 1
  double[][] m = MatrixAdd(m0, m1); // d x 1
  m = MatrixProduct(m, 0.5); // d x 1
  double[][] tc = MatrixProduct(wT, m); // ((1xd)(dx1) = 1 x 1
  double[][] wTx = MatrixProduct(wT, x); // (1xd)(dx1) = 1 x 1
  if (wTx[0][0] > tc[0][0]) return 0; else return 1;
}
```

Method Prediction computes the transpose of w so it can be used in matrix multiplication. The means of the two classes are calculated, then the average of those two means is calculated by multiplying each component value by 0.5 and then stored in matrix m.

Matrix tc is the threshold constant and is the product of the transpose of the discriminate vector, wT, and the average of the class means, m. Matrix tc will always be a 1x1 matrix, holding a single value. The value in matrix tc represents the projection of the mean onto the discriminate vector.

The projection of the item to predict, x, onto the discriminate vector is computed similarly, as the matrix product of the transpose of the discriminate vector and x. Unfortunately, because the projection of the discriminate can be positive or negative, the Boolean comparison operator to use, less-than or greater-than, will vary from problem to problem. The simplest approach is to try greater-than and see if it gives results that make sense. You can programmatically determine which operator to use, but that approach adds a lot more code than you might expect.

An option when making a prediction is to adjust the projection of the average of the class means by taking into account the probabilities of each class. This adjustment factor is log(p0 / p1), where p0 is the probability of class 0 and p1 is the probability of class 1. For the demo data, p0 = 3/8 = 0.375 and p1 = 5/8 = 0.625, so the adjustment factor is log(0.375 / 0.625) = log(0.6) = -0.22. Notice that if the two probabilities are assumed to be equal, then p0 = p1 and the adjustment factor would be log(1) = 0.

Comments and Limitations

There are actually several different variations of LDA. The one presented in this article is usually called Fisher's LDA. LDA can also be used for feature selection (identifying which predictor variables are most useful so that less-useful predictors can be ignored), as well as classification. And note there's an entirely different statistical LDA (latent Dirichlet allocation) used in natural language processing.

Although LDA binary classification is mathematically elegant, it has several critical limitations. Behind the scenes, different versions of LDA make various assumptions, for example that the predictor variables are normally distributed and have similar covariances. In many situations, these assumptions are not true. Even so, in practice, LDA often works quite well even when the math assumptions aren't met. Another math limitation is that LDA must compute the inverse of the scatter-within matrix. Matrix inversion is a very tricky process and can easily fail.

> Perhaps the biggest limitation of LDA binary classification is that it assumes the two classes are linearly separable.

Perhaps the biggest limitation of LDA binary classification is that it assumes the two classes are linearly separable. Loosely speaking, this means that when graphed as in **Figure 2**, it's possible to find a straight line that separates the two classes. The data for many problems just isn't linearly separable.

In my opinion, LDA binary classification is beautiful, but there are alternative classification algorithms, notably logistic regression classification and neural network classification, which are more practical. But LDA is sure interesting. ■

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 experts at Microsoft Research for reviewing this article: *Madison Minsk and Amy Shan*

TED NEWARD

How To Be MEAN: Express Install

The MongoDB, Express, AngularJS, Node.js (MEAN) stack is an alternative "Web stack" (some say complementary, others say supplementary) to the ASP.NET stack to which .NET developers are accustomed. This installment marks the third in the series in which I'll cover the Express library that handles HTTP processing on the server.

In the last installment (msdn.com/magazine/mt422588), I showed how to install a Node.js application into Microsoft Azure. Because doing this is essentially just committing to a Git repository (and then pushing those commits to the Azure remote repository), I'll leave those out in this and future columns, at least until I start talking to other services (such as MongoDB) on the Azure platform. Alternatively, you can run all the examples on local machines in order to follow along. Azure certainly isn't required (at least, not until I start talking about pushing to production).

Get on the Express

All train puns aside, Express is a fairly straightforward library. It's easy to work with, once you embrace "the Node.js way." Just to keep the cognitive load light, I'll start from scratch (relatively speaking). I'll assume you have a brand-new Azure site (the results of a successful "azure site create—git" command) and a simple Express application up and running.

In the first installment (msdn.com/magazine/mt185576), I mentioned npm, the Node Package Manager. This is the library and dependency manager for all Node applications. It's similar to the role NuGet plays in the Microsoft .NET Framework world. In fact, both were inspired by Ruby gems, so they share a number of the same characteristics.

As it turns out, for all of its information, npm depends on a JSON file that contains all of the dependencies—production and development-only packages—on a single file called package.json. So a logical first step is to create the file with Express as a dependency, as shown in **Figure 1**.

Figure 1 **Create a File with Express as a Dependency**

```
{
  "name": "MSDN-MEAN",
  "version": "0.0.1",
  "description": "Testing out building a MEAN app from scratch",

  "main": "app.js",

  "scripts": {},

  "author": "Ted Neward",

  "license": "ISC",

  "dependencies": {
    "express": "^4.9.8",
    "lodash": "^2.4.1",
    "serve-static": "^1.7.0"
  },

  "devDependencies": {
  }
}
```

> Express is a fairly straightforward library. It's easy to work with, once you embrace "the Node.js way."

The contents of this file are mostly self-explanatory, but there are a few things worth pointing out. First, you'd often maintain this file by hand (although there are some tools that will manage it for you). That's in keeping with the "text editor and command line" mentality of the Node.js world. As a result, would-be Node developers should be comfortable editing by hand, regardless of the availability of any tools. Second, there are two sets of dependencies here. The dependencies are the packages the npm should install in a production-type environment. The devDependencies are packages used solely by developers.

For now, the easiest way to think about this is dependencies will be installed when the app is pushed (through Git) to Azure. The devDependencies are installed (along with the contents of dependencies) when installing packages on a developer laptop, as shown in **Figure 2**.

Speaking of which, you should do that now. With just the package.json file in the current directory, enter "npm install" at the command line and watch npm pull down express, lodash (a handy library of functions and methods, including some functional tools like map and filter, as well as extensions to arrays and objects), and serve-static (which I'll discuss later) onto your local machine.

Like NuGet, npm installs not just the package, but all its dependencies, as well. This is what causes the tree display in the terminal. For large umbrella packages that incorporate a large number of popular packages, this display can get quite long. The npm tool also notes

SharePoint LIVE!
TRAINING FOR COLLABORATION

ORLANDO
ROYAL PACIFIC RESORT AT
UNIVERSAL ORLANDO

NOV 16-20

LIVE! 360
TECH EVENTS WITH PERSPECTIVE

5 Great Conferences 1 Great Price

Charting Collaboration

Chart the best collaboration course with SharePoint Live!, providing leading-edge knowledge and training for SharePoint administrators, developers, and planners who must customize, deploy and maintain SharePoint Server on-premises and in Office 365 to maximize the business value.

Whether you are a Manager, IT Pro, DBA, or Developer, SharePoint Live! brings together the best the industry has to offer for 5 days of workshops, keynotes, and sessions to help you work through your most pressing SharePoint projects.

SQL Server LIVE!
SQL SERVER FOR MODERN DEVELOPERS

Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS

SharePoint LIVE!
TRAINING FOR COLLABORATION

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

TECHMENTOR
IN-DEPTH TRAINING FOR IT PROS

REGISTER BY OCTOBER 14 AND **SAVE $300!**

Use promo code SPOCT1

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

SUPPORTED BY
SharePoint msdn magazine Redmond MAGAZINE

PRODUCED BY
1105 MEDIA
YOUR GROWTH. OUR BUSINESS.

SPLIVE360.COM

that I've left out two entries in my package.json: the repository field where this package comes from and the README data. These are used mostly for npm packages installed into the central npm library. I don't find them particularly appropriate for an application, but there's certainly no harm in having them if you prefer warning-less npm activity.

As this series continues, I will add additional packages to package.json, so you should periodically run "npm install" to get the new packages. As a matter of fact, notice in the install output tree for express, the debug package (which I used in the last installment) is a dependency, but I didn't put it into the package.json file directly. It's been my habit that my package.json file should explicitly list all dependencies I use directly (as opposed to those used by the libraries, but never called directly), so go ahead and add it to package.json and do another "npm install":

```
"dependencies": {
  "debug": "^2.2.0",
  "express": "^4.9.8",
  "lodash": "^2.4.1",
  "serve-static": "^1.7.0"
},
```

This way I know which version of debug I'm actually using (as opposed to what version Express is using), in case that difference becomes important. By the way, the traditional way to install a library (now that you know what the file format looks like) is to use "npm install <package> --save." Using the "--save" argument causes npm to modify the package.json file (as opposed to just installing it without record).

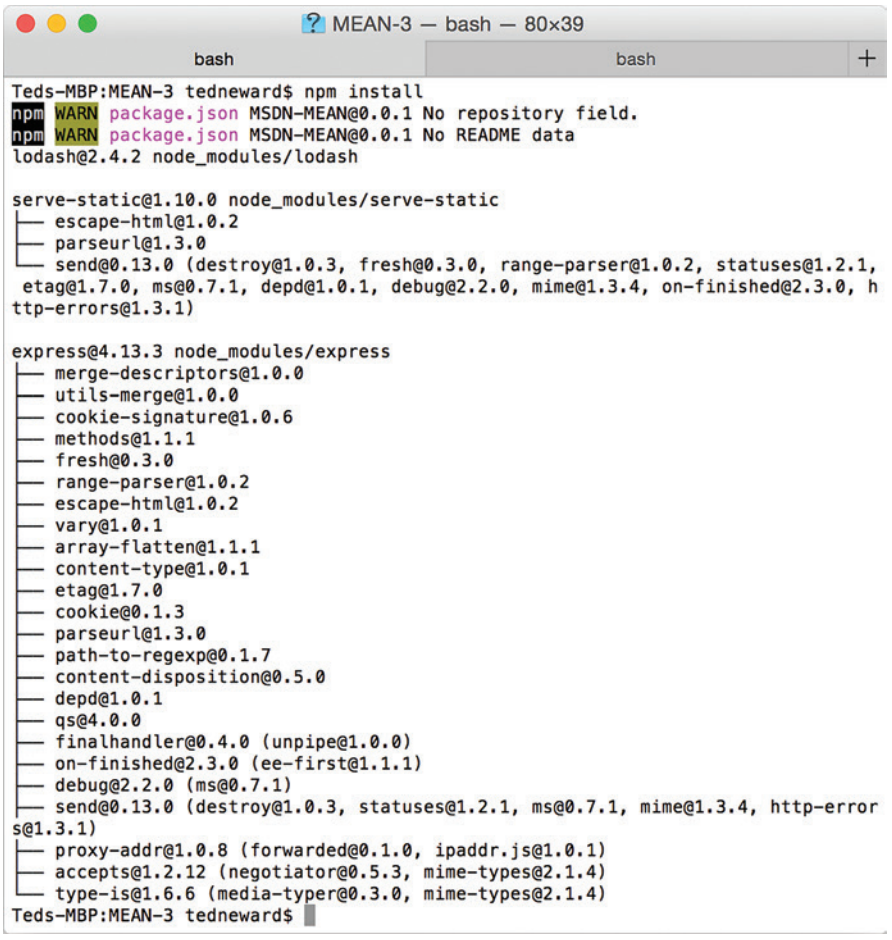


Figure 2 **Install Your Dependencies**

> Like NuGet, npm installs not just the package, but all its dependencies, as well.

In fact, if you do an "npm install express --save," npm will update the express entry to the latest version (which, as of this writing, is 4.13.3), even though it's already there. There's yet a third way, using another package called yeoman to scaffold out an Express app, but part of the goal here is to understand all the moving parts in MEAN, so I'll leave that for later.

Hello, Express

Once the Express bits are installed, it's a simple matter to write the ubiquitous "hello world" in Express. In a file called app.js (because that's what package.json has as its entry for "main"), the code in **Figure 3** provides the simple-yet-necessary homage to the Gods of Computer Science.

The first two lines are the require calls you saw in the previous installment, which is how Node.js loads modules at the moment. This will change in ECMAScript 6, once that's ratified and shipped.

Figure 3 **The Code for the Hello World Express**

```
// Load modules
var express = require('express');
var debug = require('debug')('app');

// Create express instance
var app = express();

// Set up a simple route
app.get('/', function (req, res) {
  debug("/ requested");
  res.send('Hello World!');
});

// Start the server
var port = process.env.PORT || 3000;
debug("We picked up",port,"for the port");
var server = app.listen(port, function () {

  var host = server.address().address;
  var port = server.address().port;

  console.log('Example app listening at http://%s:%s', host, port);

});
```

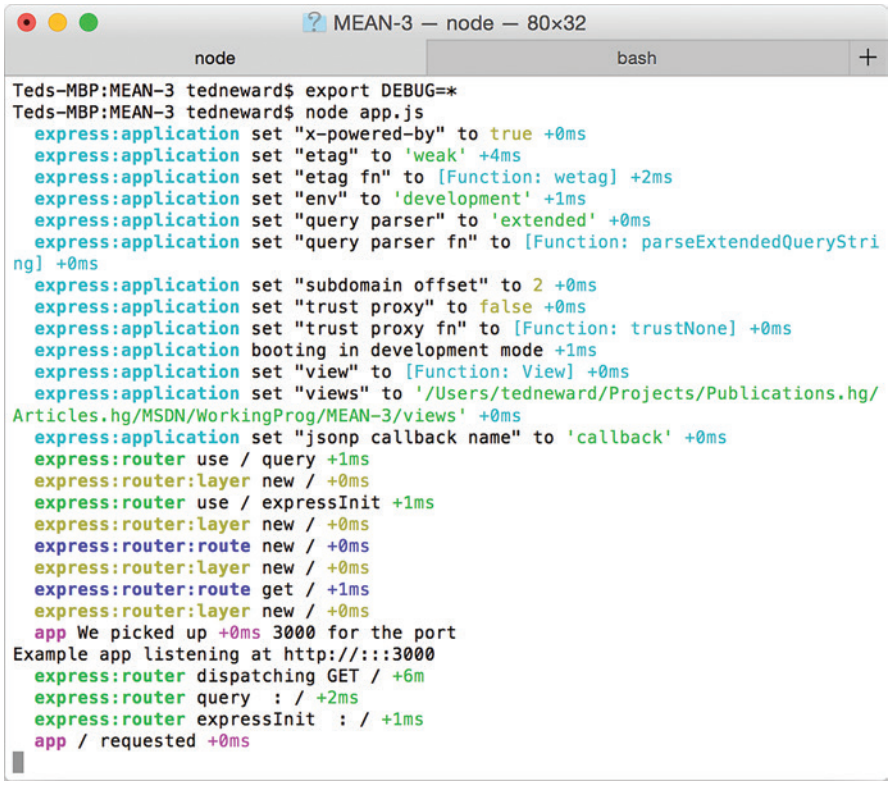


Figure 4 **Express with Debug Logging**

These give you access to the Express and debug packages, respectively. The Express module fetched by require is actually a function that, when called, yields the Express app itself—hence, it goes into app (by convention).

The next line is what Express calls a route. This is similar in many respects to the routing tables more popular in ASP.NET and ASP.NET MVC of late. This maps a request verb (get) with a relative URL path ("/" in this case) to a function. In this case, it's an anonymous function literal that responds with "Hello World." Note the function also uses the debug object to spew out a quick line after receiving a request.

> The Express module fetched by require is actually a function that, when called, yields the Express app itself—hence, it goes into app (by convention).

Last, the code checks the current environment to see if there's a PORT environment variable specified (which there will be in Azure). If not, it assigns 3000 to port. Then the app object is told to listen, which is a blocking call. It will execute the anonymous function literal passed in when it begins listening. Then the process will just wait for an incoming request. Assuming the request is for "/," it replies with "Hello World."

Debug, Express and You

As mentioned in the previous installment, the debug output doesn't show up unless the DEBUG environment variable is set to the same string used in the require step. In this case, that's app. Express also uses debug, and if DEBUG is set to "*," then all of the Express diagnostic output will also display. It's probably a bit too voluminous to use for debugging on a regular basis, but it can be helpful to see it scroll by in the early days of working with Express. That will help give you a sense of the various parts and what's being invoked when. There's an example of its output (with a few requests) shown in **Figure 4**.

> In many ways, Express is a lot like ASP.NET.

To turn debugging off, simply set DEBUG to nothing. To view more than one debug stream, but not all of them, just comma-separate the names:

```
DEBUG=app,express:router,express:router:layer
```

That command will only show output from those three debug streams.

Wrapping Up

In many ways, Express is a lot like ASP.NET. That's true not only in how it handles HTTP traffic, but how it serves as a sort of "central hub" around which dozens (if not hundreds) of other packages depend and extend. The "serve-static" package, for example, gives an Express app a pre-built way to serve up a directory for static (that is, "not executing on the server side") assets, like images and fonts. In the next installment, I'll start looking at how to use the Express request and response objects. (If you can't wait, check out the Express documentation at expressjs.com.) In the meantime … happy coding! ■

Ted Neward *is the CTO at iTrellis, a consulting services company. He has written more than 100 articles and authored or co-authored a dozen books, including "Professional F# 2.0" (Wrox, 2010). He is 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.*

Thanks to the following technical expert for reviewing this article: *Shawn Wildermuth*

Anachronisms

We geeks pride ourselves on keeping current with advancing technology. We upgrade to the latest phones, download the latest apps, sling the latest slang; with automatic updates to ensure we never fall behind. But if we look at the apps that we're writing, we'll see ourselves reaching into the past to communicate with users. We can't or won't rid our programs of anachronisms.

An anachronism, according to dictionary.com, is: "… a thing belonging or appropriate to a period other than that in which it exists, especially a thing that is conspicuously old-fashioned." The classic example comes from Shakespeare's "Julius Caesar," where Cassius says: "The clock has stricken three." The Romans never developed striking clocks, but I suppose "The sundial says three" wouldn't have worked, especially in cloudy England.

Anachronisms often take the form of pictures. Windows 10, not a month old as I write these words, uses a picture of a floppy disk on the Save button. When was the last time you even *saw* an actual floppy disk, never mind used one? (Although … the U.S. Air Force still uses them in their missile silos, see slate.me/1LOAwzR. I don't know whether to be happy or sad about this.) USB sticks blew them out of the water a decade ago, but that's still the picture we use to convey to our user the idea of saving.

Sometimes anachronisms take the form of outdated phrases. Consider the term "disk drive." The first word indicates a circular shape, while the second word implies some sort of motion. Neither of these applies to a modern solid-state disk drive, which neither is circular nor moves.

Sounds can be anachronistic, too. Consider the "ka-ching" sound you hear when you enter a check into Quicken. It harks back to an old-fashioned, mechanical cash register, which I haven't seen in quite some time. I wonder how long before the picture of the check itself becomes an anachronism.

Or consider the modern smartphone camera, which emits the "gshpratz" sound (I love onomatopoeia, don't you?) of a mechanical shutter every time it snaps a photo. I wonder if that noise irritates any nearby tigers and causes fatelfies, as I wrote last month (see msdn.com/magazine/mt422590).

The latter sound, by law, can't be turned off or even changed on most phones sold in Asia today. Its unshakeable purpose is to alert potential targets of voyeuristic photos that someone nearby is snapping them. I wonder if this will lead to a new black market for silent phones. I suspect that the promulgators of this law never considered that a bad guy might stick a piece of duct tape over the phone's speaker, an excellent example of a hardware solution to a software problem. Remember, friends, when duct tape is outlawed, only outlaws will tape ducts. Red Green (redgreen.com), look out.

At what point does an anachronism lose its original meaning and take its own place in the modern lexicon? Today's digital natives (Mark Prensky's coinage, essentially anyone younger than 30) have heard Quicken's "ka-ching" far more often than from a mechanical cash register. And digital symbionts (my coinage, see msdn.com/magazine/mt147245, essentially anyone younger than 5 years old) have never heard a cash register in their lives, nor will they outside of a museum. When does a picture or term or sound lose its skeuomorphism and become just an arbitrary pattern? I guess when enough of the cohort that recognizes the original item dies, or at least leaves the user population.

Anachronisms often take the form of pictures. Windows 10, not a month old as I write these words, uses a picture of a floppy disk on the Save button. When was the last time you even *saw* an actual floppy disk, never mind used one?

I wonder which current computing technology will be the next to disappear from our actual world, maintaining a ghostly undead existence as an anachronistic icon. I can imagine Annabelle's and Lucy's children pointing at a picture on their toolbar and saying to me, "Cranky old fart Grandpa, what was a 'printer'?" ■

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.*

The reporting platform for all of your business needs

Design, publish, view, print and export operational reports such as invoices, expense reports, tax and government forms, as well as strategic and analytical reports, such as sales performance, budgeting, and revenue analysis. ActiveReports gives you the power and flexibility you need to turn your data into informative pixel-perfect reports across the enterprise.

15+ years on the market	**300,000+** developer community	**Multi-platform** report viewers	**Trusted** by a worldwide customer base	**Easy** licensing

Download your free trial at
activereports.grapecity.com

GrapeCity ActiveReports