THE MICROSOFT JOURNAL FOR DEVELOPERS

MARCH 2016 VOL 31 NO 3

msdn magazine

THE MICROSOFT JOURNAL FOR DEVELOPERS

MARCH 2016 VOL 31 NO 3

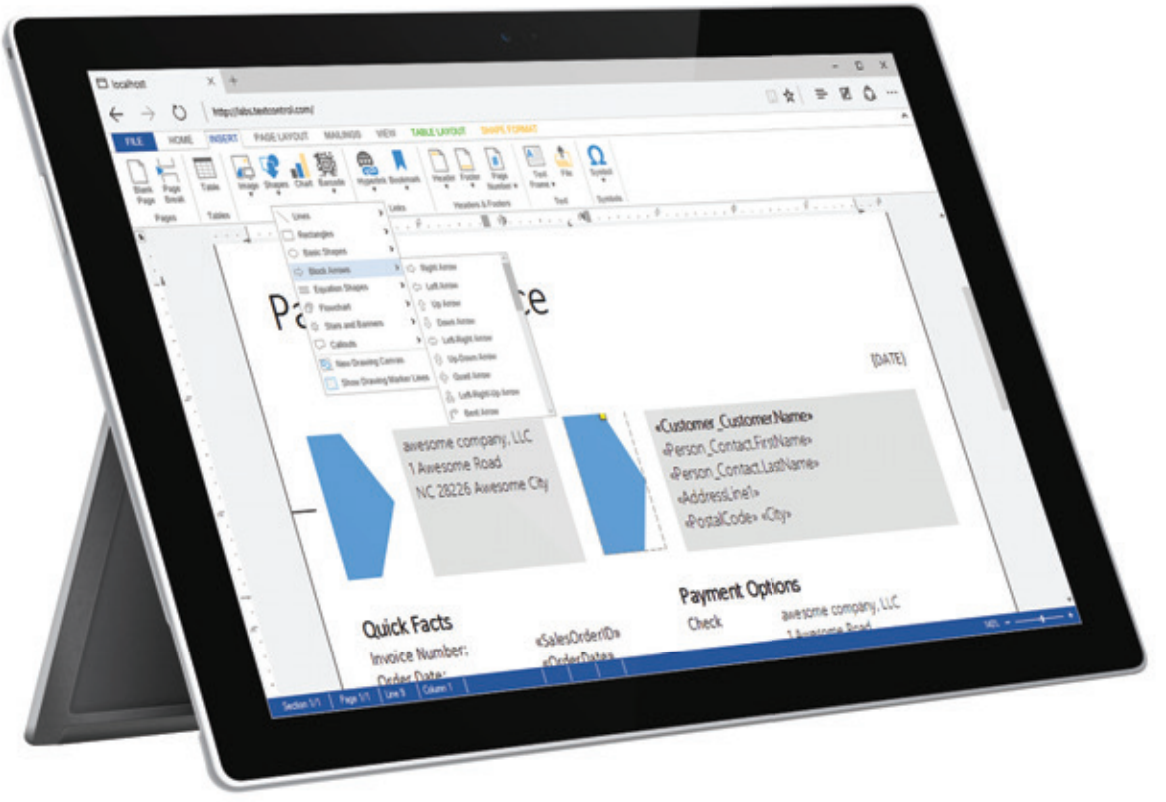
msdn magazine

COLUMNS

Microsoft

TEXT CONTROL

ALL ABOUT DOCUMENTS



Text Control provides deep functionality components and consulting to integrate reporting and word processing into .NET Windows, web and mobile applications.

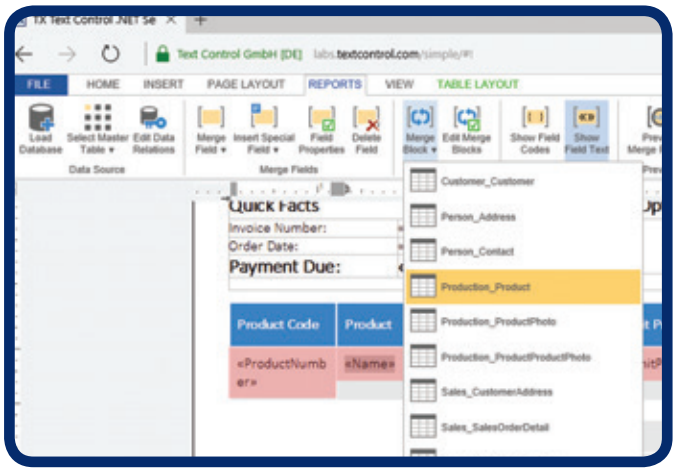
Visual Studio Partner

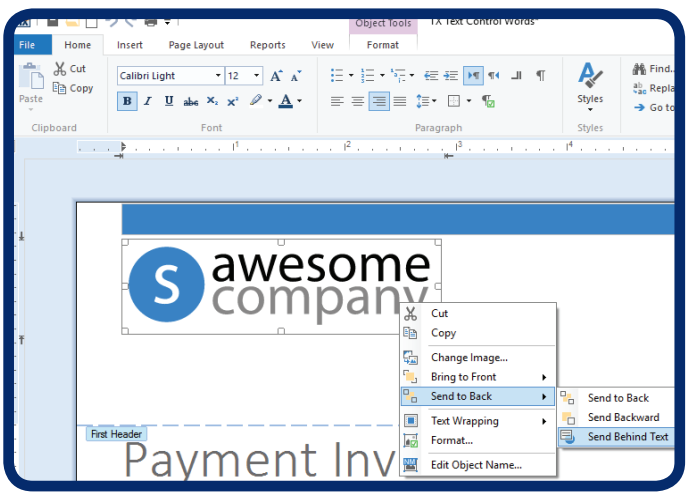
Call us: 855-533-TEXT

Reporting and Mail Merge

The Text Control Reporting Framework combines powerful reporting features with an easy-to-use, MS Word compatible word processor for ASP.NET, WPF and Windows Forms. Users can create documents and templates using ordinary Microsoft 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.





MS Word compatible Rich Text Editing

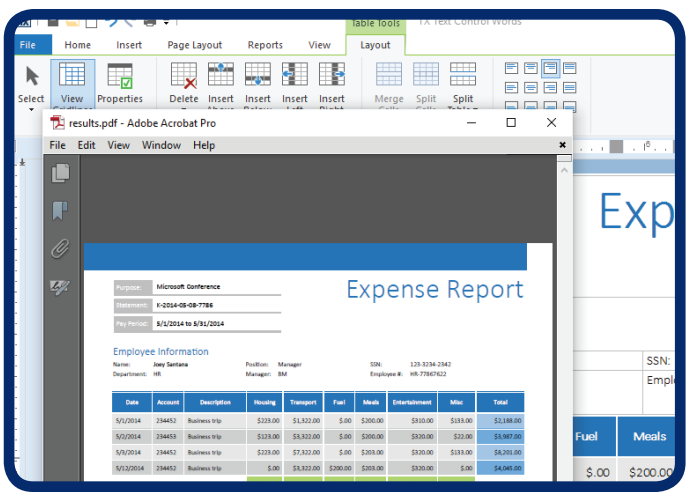
These feature-complete, fully programmable rich edit controls offer developers a broad range of word processing features in reusable UI and non-UI components for ASP.NET, WPF and Windows Forms.

They provide comprehensive text formatting, powerful mail merge features and all word processing key concepts such as table support, images, headers and footers, page sections and spell checking.

Load and Save Adobe PDFs

Using TX Text Control, Adobe PDF and PDF/A documents can be created including document access security settings and digital signatures.

PDF documents can be imported to be viewed, edited and converted. Edit PDF documents just like any other format such as DOC or DOCX. A fully featured API can be used to modify the content or to search through the document.



Live demo and 30-day trial version download at:

www.textcontrol.com

X13 released

Software ▪ Training ▪ Consulting

TEXT CONTROL

msdn magazine

MARCH 2016 VOLUME 31 NUMBER 3

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/QuickQote.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.

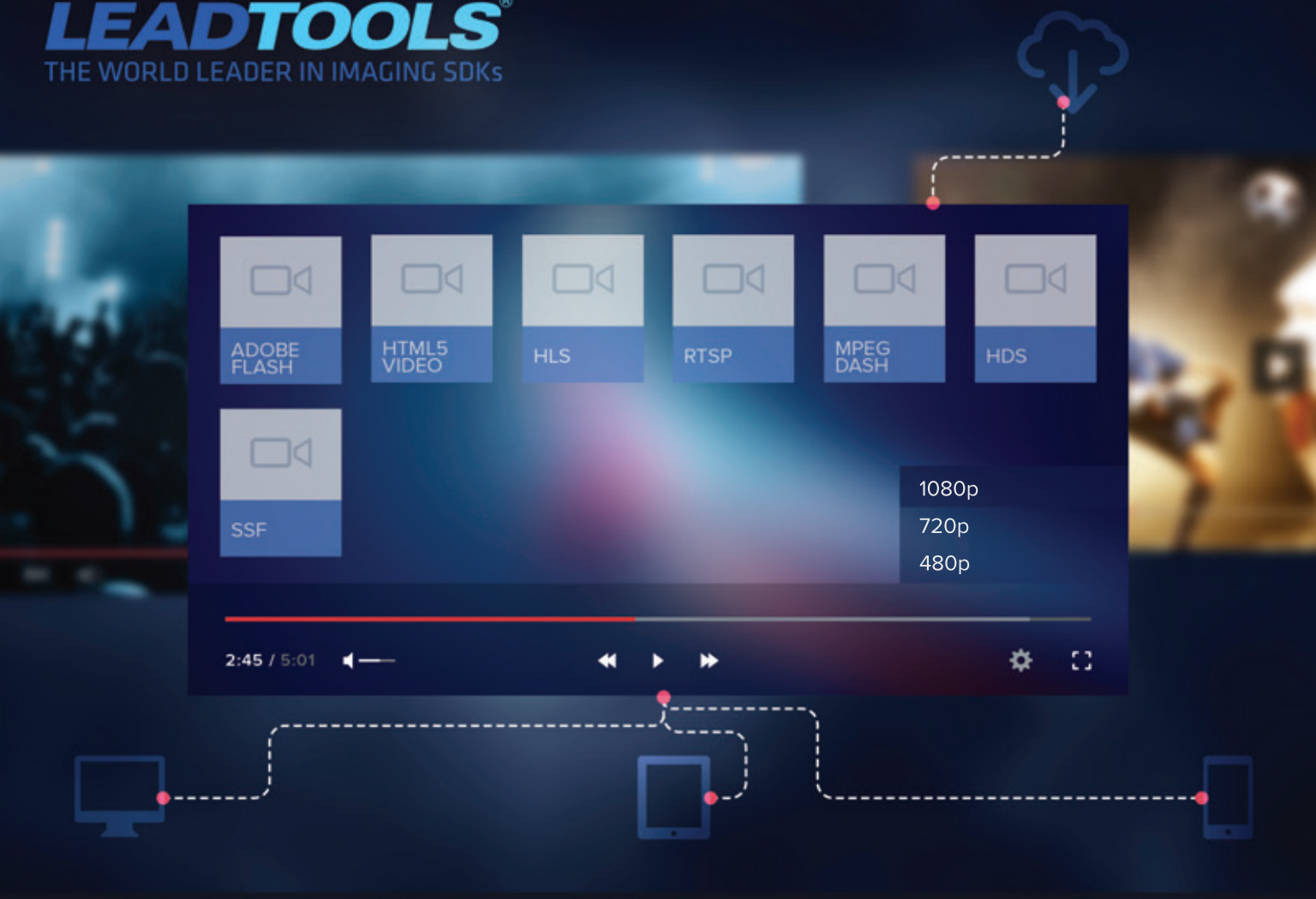
Microsoft

BPA WORLDWIDE

1906 2006 100 AMERICAN BUSINESS MEDIA

WPA Western Publications Association

LEADTOOLS®
THE WORLD LEADER IN IMAGING SDKs



TRANSCODE MULTIMEDIA FILES AND LIVE STREAMS FOR REAL-TIME PLAYBACK

Media Streaming Server SDK

High-level media streaming server framework with minimal coding required

Stream media from files, cameras, HDTV devices, TCP and UDP streams for playback on any client

Restream live IP audio/video feeds from RTSP and ONVIF devices

Automatically transcode on the fly for clients requiring different protocols

DOWNLOAD OUR 60 DAY EVALUATION
WWW.LEADTOOLS.COM

LEAD TECHNOLOGIES

SALES@LEADTOOLS.COM
800.637.1840

MICHAEL DESMOND

Chasing Fogbank

In 2000 the National Nuclear Security Administration (NNSA) embarked on a program to extend the life of select nuclear warheads—among them, the W76 model manufactured between 1978 and 1987. Used in Trident submarine launched ballistic missiles, the 100-kiloton warheads were among the most numerous in the U.S. stockpile.

A problem soon developed. Engineers working on the project found that an important component of the W76 bomb design—a classified material known as "Fogbank"—could not be manufactured. The facility that had produced Fogbank had been shuttered, and documentation about its manufacture was either lost or incomplete. Worse still, the staff that worked to create Fogbank in the 1980s was no longer with the government.

Engineers working on the project found that an important component of the W76 bomb design—a classified material known as "Fogbank"—could not be manufactured.

Dennis Ruddy, a general manager working on the refurbishment effort at the time, described Fogbank thusly: "The material is classified. Its composition is classified. Its use in the weapon is classified, and the process itself is classified."

Fogbank may be an interstage material that helps trigger a fusion explosion by mediating the pulse of X-rays produced in the initial fission stage of a thermonuclear bomb. Regardless, the secrecy around it was a big part of the problem. Hobbled by institutional amnesia, parts of the U.S. nuclear refurbishment effort ground to a halt.

That left two solutions: Reverse engineer the manufacturing process from extant material, documents and facilities, or concoct a wholly new material to replace Fogbank. The government opted for both, throwing time, staff and money at the problem until it went away. Fogbank was eventually re-engineered. Despite years of delay and expenditures in the tens of millions of dollars, the W76 refurbishment program pushed forward.

Incidents like these provide a cautionary tale for software developers, who are often challenged to refurbish aging assets of their own. What happens in the software world when institutional memory falters? The global Y2K remediation effort at the turn of the century offered a glimpse, as aging Cobol and Fortran coders shuffled out of retirement to work on billions of lines of mostly forgotten code.

In fact, dangers arise whenever the work stops. Vital contextual knowledge about a software project begins to erode almost immediately, especially as team members disperse. Special "revival documentation" with system setup and configuration information can help address key gaps, but even a major investment in documentation can't catch everything.

The Fogbank effort illustrated this. For months, engineers struggled to understand why the material they had reverse engineered wasn't performing optimally. It was later learned that the modern cleaning processes used to purify component materials didn't exist when Fogbank was first manufactured in the 1980s. In short, the end product they were producing was *too* pure. Only by adding specific impurities back into the process was the team finally able to fully reverse engineer Fogbank.

Have you faced your own Fogbank struggle? Let me know how you managed it. E-mail me at mmeditor@microsoft.com.

Switch to Amyuni PDF

AMYUNI

Create and Edit PDFs in .NET, COM/ActiveX, WinRT & UWP

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

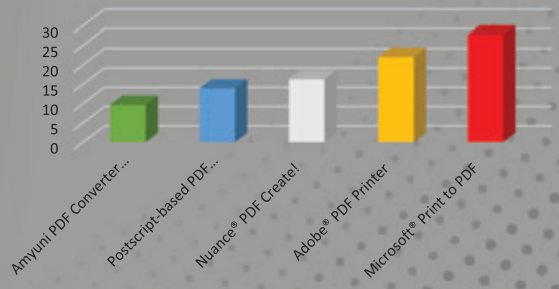
Other Developer Components from Amyuni®

- 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

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 Gift of Anger

The first time I sent off an angry e-mail to a colleague, I groveled and apologized afterward in person and over e-mail. Gradually, I got really good at these groveling apologies. The pattern was often the same: a teammate would offer an opinion that I thought was silly, ill-informed or dangerous. Riled up by their "stupidity," I would slay their arguments. Inevitably, I'd overdo it, and then regret it all.

Yet, no matter how many times I did this, I could not stop myself! No amount of tactical moves, such as setting an automatic delay for sending e-mails, really fixed the root of the problem. But recently, a series of revelations helped create a shift in my behaviors.

Judging Stupidity

I discovered that I was most offended by "stupidity." My anger was often aimed at my coworker being ignorant in some way. I began to realize that deep down maybe, just *maybe*, I felt stupid myself. I'd never acknowledged that parts of me could be dim or naive.

But then I asked myself, "What if I *were* stupid? What is the gift in being half-witted?" A few answers immediately came to mind: I'd break rules (because I wouldn't even know they existed); I'd say what was on my mind, regardless of what others thought (because I'd be unaware of wanting to please others); I'd do what I wanted on every project (I wouldn't care for any standards to meet); and I'd be spontaneous (I wouldn't be "smart" enough to analyze every decision.)

I realized that what made me feel stupid was *exactly* what made me brilliant. I was trying to have only the brilliant moments in my life, without allowing the moments that reminded me how helpless and dense I could be. I discovered that without my dullness, I couldn't be sharp.

Because I refused to acknowledge the foolish parts of me, the universe—playfully—kept bringing people into my life who I'd judge as dummies. The only way I could acknowledge, honor, cherish, and love *all* of myself—because I couldn't see it in myself—was to see it in others and learn to love them for it.

Without loving my ordinary and imperfect parts, I couldn't fully appreciate my clever and funny parts. If I suppressed my darkness, my light couldn't shine. In loving my darkness, I unleashed my light.

Talk Is Cheap ... or Is It?

People who spoke too much at meetings were another anger flash point for me. I prided myself on speaking only when I had deep, incisive insights to offer, and resented colleagues who were not as efficient as I was.

In a flash of insight, I realized that deep down, what led anyone to speak at meetings was a desire to be loved. It takes courage and vulnerability—even for my chatty friends, even if they don't realize it—to put themselves on the spot. Even I, with my occasional incisive insight, was ultimately looking for love from my colleagues. I could fool myself by describing my needs as a desire for respect, admiration or affirmation, but those are really just code words for love.

When I speak at meetings, I am once again a little kid sitting at a table, telling a story, wanting to be listened to and encouraged. Seen from this perspective, I could no longer judge my chatty coworkers. If I were looking for love, I'd want to be loved. So why not give that love to my associates when they were looking for it?

In a flash of insight, I realized that deep down, what led anyone to speak at meetings was a desire to be loved.

Today, when a colleague gets overly chatty at meetings, I play with giving him full attention, asking him follow-up questions, or acknowledging the positives in his statements, especially when I want to disagree most vehemently or run away. As I do this, I begin to lose my own inhibitions and insecurities about being judged for my faults. The non-judgment I show to others comes back to me a hundred-fold.

Every interaction I have with the world is either a request for love, or a request to be loved. When I help a coworker or mentor a junior employee, I'm loving. When I'm nervously walking to my performance review, I'm looking for love.

Love and Anger

In dealing with anger at work, I've realized that difficult coworkers are gifts from providence to deepen my own personal and spiritual growth. When I get angry at someone else, I'm really getting angry at myself. When I shout at someone, I'm actually shouting at myself. Everybody I talk to is a mirage; I'm always just talking to myself—past, present, future, unacknowledged or healed versions of myself.

The upshot: To paraphrase author Debbie Ford, everyone I have a "problem" with is either someone I can help by my example, or someone who I can learn to treat with love and compassion.

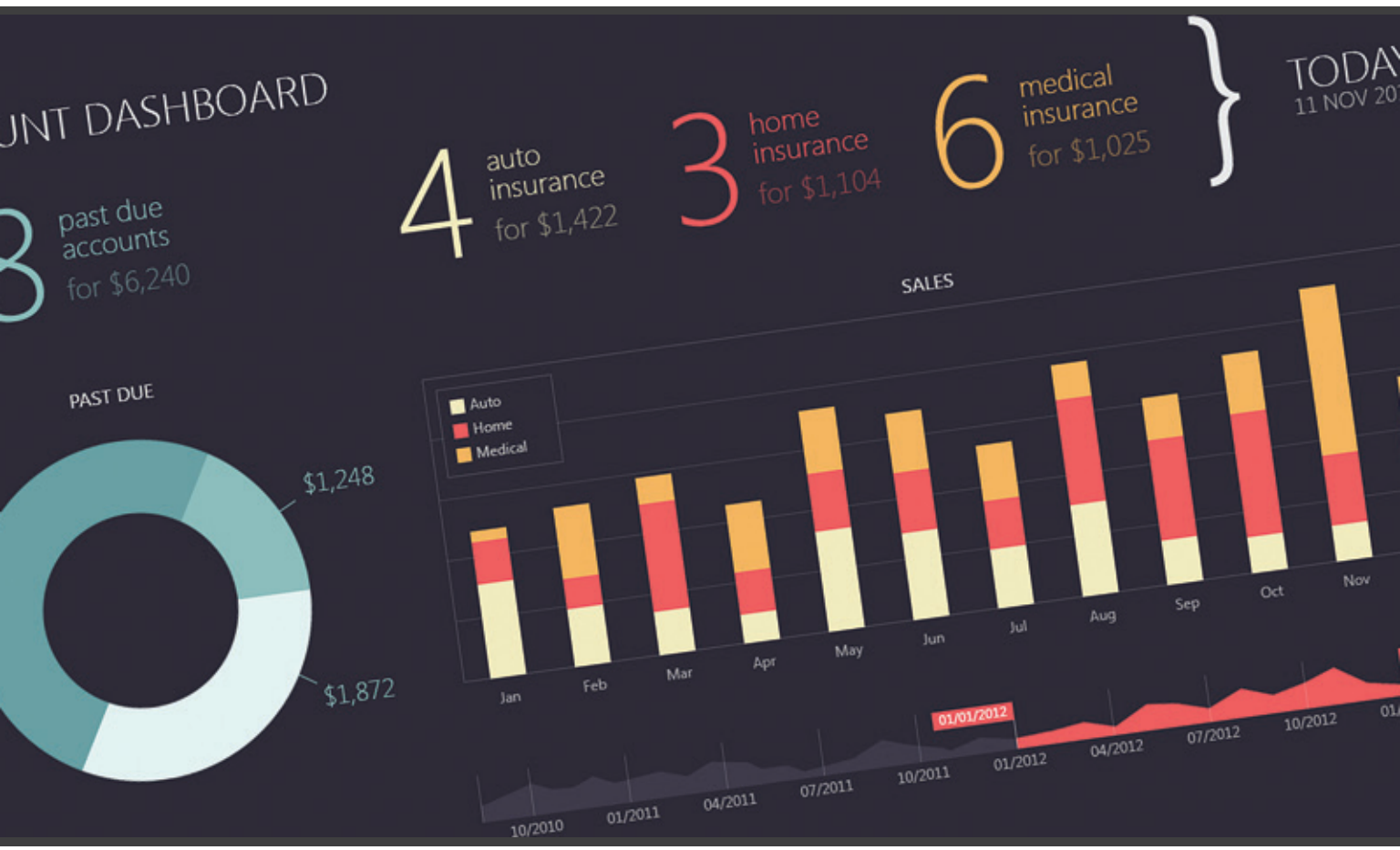
In learning to get along with others—especially those who are difficult—I finally, at long last, have learned to love myself. ■

KRISHNAN RANGACHARI *is a career coach for hackers. Visit radicalshifts.com to download his free career success kit.*

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

The Query Stack of a CQRS Architecture

Today, Command and Query Responsibility Segregation (CQRS) is one of the architectural topics most talked about. At its core, CQRS is nothing more than common sense and all it recommends is that you code the queries and commands that your system requires through distinct and ad hoc stacks. You might want to have a domain layer in the command stack and organize business tasks in domain services and pack business rules in domain objects. The persistence layer can also be coded in total freedom just picking the technology and the approach that best fits your needs, whether plain relational tables, NoSQL or event stores.

What about the read stack, instead? The good news is that once you start looking at a CQRS architecture, often you don't feel the need to have a distinct section of the domain layer for just reading data. Plain queries out of ready-made tables are all you need. The query stack performs read-only operations against the back end that don't alter the state of the system. Because of this, you might not need any form of business rules and logical intermediation between presentation and storage—or, at least nothing beyond the basic capabilities of SQL advanced query operators such as GROUP BY, JOIN and ORDER. In the implementation of a modern query stack, the LINQ language backed in the Microsoft .NET Framework is immensely helpful. In the rest of this column, I'll go through a possible implementation of a read stack where the storage is designed to be close to the organization of data required by the presentation.

Having Ready-Made Data

In a CQRS project, you typically plan separate projects for the query and command stacks and you can even have different teams taking care of each stack. A read stack essentially consists of two components: a collection of Data Transfer Objects (DTOs) to deliver data up to the presentation layer and a database context to perform physical reads and populate DTOs.

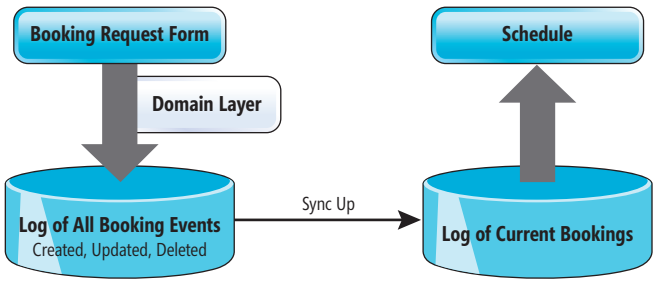


Figure 1 **Booking Schema in a CQRS Architecture**

To keep the query stack as lean and mean as possible, you should aim at having a close match between the data stored and data to present. Especially these days, this might not be the case: It's more common the scenario in which data is ideally stored in a given format, but must be arranged in a significantly different format to be effectively consumed. As an example, think of a software system to book meeting rooms in a business environment. **Figure 1** gives a graphical perspective of this scenario.

The user is offered a few forms to create a booking request or update an existing booking. Each action is logged in the command data store as is. Scrolling through the list of logged booking events, one can easily track when each booking was entered into the system, when it was modified, how many times it was modified and if and when it was deleted. This is definitely the most effective way to store information about a dynamic system where the state of stored items might change over time.

Saving the state of the system in the form of events has many benefits as summarized in my August 2015 Cutting Edge column (msdn.com/magazine/mt185569), but it can't offer an immediate view of the current state of the system. In other words, you might be able to dig out the full history of a single booking, but you're not immediately able to return the list of pending bookings. Users of the system, instead, are also typically interested in getting the list of bookings. This creates the need for two distinct stores kept in sync. Every time a new event is logged, the state of the involved entities should be updated (synchronously or asynchronously) for easy queries.

During the synchronization step, you make sure that useful data is extracted from the log of events and massaged into a format that's easy to consume from the UI through the query stack. At this point, all the query stack needs to do is shape up data for the particular data model of the current view.

More on the Role of Events

A common objection is, "Why should I save data in the form of events? Why not simply save data in the form in which it will be used?"

The answer is the classic, "It depends," and the interesting thing is that it doesn't typically depend on you, the architect. It depends on the business needs. If it's key for the customer to track the history of a business item (that is, the booking) or to see what the list of bookings was on a given day and if the availability of rooms may change over time, the simplest way to solve the problem is by logging events and building any required data projection on top of that.

By the way, this is also the internal mechanics used by business intelligence (BI) services and applications. By using events, you can even lay the ground for some in-house BI.

Read-Only Entity Framework Context

Entity Framework is a common way to access stored data, at least from within .NET and ASP.NET applications. Entity Framework funnels database calls through an instance of the DbContext class. The DbContext class provides read/write access to the underlying database. If you make the DbContext instance visible from the uppermost layers, you expose your architecture to the risk of also receiving state updates from within the query stack. It's not a bug, per se; but it's a serious violation of architectural rules that you want to avoid. To avoid write access to the database, you might want to wrap the DbContext instance in a container and disposable class, as shown in **Figure 2**.

You can use Database wherever you'd like to use a DbContext only to query data. Two aspects make the Database class different from a plain DbContext. First and foremost, the Database class encapsulates a DbContext instance as a private member. Second, the Database class exposes all or some of the DbContext collections as IQueryable<T> collections rather than as DbSet<T> collections. This is the trick to enjoy the query power of LINQ while being unable to add, delete or just save changes back.

Shaping Up Data for the View

In a CQRS architecture, there's no canonical standing as far as the organization of the application layer is concerned. It can be distinct for the command and query stacks or it can be a unique layer. Most of the time, the decision is left to the vision of the architect. If you managed to keep dual storage and therefore have data made to measure for presentation, then you nearly have no need of any complex data retrieval logic. Subsequently, the implementation of the query stack can be minimal. In the query portion of the application layer code, you can have direct data access code and call directly the Entity Framework DbContext entry point. However, to really keep things limited to query operations, you just use the Database class instead of the native DbContext. Anything you get back from the Database class is then only queryable via LINQ, though. **Figure 3** provides an example.

Figure 2 **Wrapping the DbContext Instance in a Container and Disposable Class**

```
public class Database : IDisposable
{
  private readonly SomeDbContext _context = new SomeDbContext();

  public IQueryable<YourEntity> YourEntities
  {
    get
    {
      return _context.YourEntities;
    }
  }

  public void Dispose()
  {
    _context.Dispose();
  }
}
```

As you can see, the actual projection of data you would get from the query is specified at the last minute, right in the application layer. This means you can create the IQueryable object somewhere in the infrastructure layer and move it around across the layers. Each layer has the chance to further modify the object refining the query without actually running it. By doing so, you don't need to create tons of transfer objects to move data across layers.

Let's consider a fairly complex query. Say, for example, that for one of the use cases in the application you need to retrieve all invoices of a business unit that haven't been paid 30 days after due payment terms. If the query expressed a stable and consolidated business need, not subject to further adaptation along the way, it wouldn't be such a tough query to write in plain T-SQL. The query is made of three main parts: get all invoices, select those specific of a given business unit and select those that haven't been paid yet after 30 days. From an implementation perspective, it makes no sense at all to split the original query in three sub queries and do most of the work in memory. Yet, from a conceptual perspective, splitting the query in three parts makes it far easier to understand, especially for domain newbies.

LINQ is the magic tool that lets you express the query conceptually and have it executed in a single step, with the underlying LINQ provider taking care of translating it into the proper query language. Here's a possible way to express the LINQ query:

```
var queryable = from i in db.Invoices
                              .Include("Customers")
  where i.BusinessUnitId == buId &&
    DateTime.Now - i.PaymentDueBy > 30
              select i;
```

The expression doesn't return data, however. The expression just returns a query that hasn't been executed yet. To execute the query and get related data, you must invoke a LINQ executor such as ToList or FirstOrDefault. Only at that point is the query built, putting together all the pieces and returning data.

LINQ and Ubiquitous Language

An IQueryable object can be modified along the way. You can do that by calling Where and Select methods programmatically. In this way, as the IQueryable object traverses the layers of your system, the final query emerges through the composition of filters. And, more important, each filter is applied only where the logic on which it's based is available.

Another relevant point to make about LINQ and IQueryable objects in the query stack of a CQRS architecture is readability

Figure 3 **A Query from the Database Class Via LINQ**

```
using (var db = new Database())
{
  var queryable = from i in db.Invoices
                              .Include("Customers")
    where i.InvoiceId == invoiceId
    select new InvoiceFoundViewModel
    {
      Id = i.InvoiceId,
      State = i.State.ToString(),
      Total = i.Total,
      Date = i.IssueDate,
      ExpiryDate = i.GetExpiryDate()
    };

  // More code here
}
```

and ubiquitous language. In domain-driven design (DDD), the ubiquitous language is the pattern that suggests you develop and maintain a vocabulary of unequivocal business terms (nouns and verbs) and, more important, reflect those terms in the actual code. A software system that implements the ubiquitous language, for example, will not delete the order but "cancel" the order and won't submit an order but "check out."

The biggest challenge of modern software isn't in technical solutions, but in deep understanding of business needs and in finding a perfect match between business needs and code. To improve the readability of queries, you can mix together IQueryable objects and C# extension methods. Here's how to rewrite the previous query to keep it a lot more readable:

```
var queryable = from i in db.Invoices
                            .Include("Customers")
                            .ForBusinessUnit(buId)
                            .Unpaid(30)
                select i;
```

ForBusinessUnit and Unpaid are two user-defined extension methods that extend the IQueryable<Invoice> type. All they do is add a WHERE clause to the definition of the ongoing query:

```
public static IQueryable<Invoice> ForBusinessUnit(
  this IQueryable<Invoice> query, int businessUnitId)
{
  var invoices =
    from i in query
    where i.BusinessUnit.OrganizationID == buId
    select i;
  return invoices;
}
```

Analogously, the Unpaid method will consist of nothing more than another WHERE clause to further restrict the data being returned. The final query is the same, but the expression of it is a lot clearer and hard to misunderstand. In other words, through the use of extension methods you nearly get a domain-specific language, which, by the way, is one of the goals of the DDD ubiquitous language.

Wrapping Up

If you compare CQRS to plain DDD, you see that in most cases you can reduce complexity of domain analysis and design by simply focusing on use cases and related models that back up actions that alter the state of the system. Everything else, namely actions that just read the current state, can be expressed via a far simpler code infrastructure and be no more complex than plain database queries.

It should also be noted that in the query stack of a CQRS architecture, sometimes even a full-fledged O/RM might be overkill. At the end of the day, all you need to do is query data, mostly from ready-made relational tables. There's no need for sophisticated things such as lazy loading, grouping, joins—all you need is already there, as you need it. A nontrivial O/RM like Entity Framework 6 might even be too much for your needs, and micro O/RMs like PetaPoco or NPoco might even be able to do the job. Interestingly, this trend is also partly reflected in the design of the new Entity Framework 7 and the new ASP.NET 5 stack. ■

Dino Esposito *is the author of "Microsoft .NET: Architecting Applications for the Enterprise" (Microsoft Press, 2014) and "Modern Web Applications with ASP.NET" (Microsoft Press, 2016). A technical evangelist for the .NET and Android platforms at JetBrains, and frequent speaker at industry events worldwide, Esposito shares his vision of software at software2cents@wordpress.com and on Twitter: @despos.*

Debugging Improvements in Visual Studio 2015

Andrew Hall

Despite our best efforts to write code that works correctly the first time, the reality is that as developers, we spend a lot of time debugging. Visual Studio 2015 brought new functionality and improvements to help developers identify issues earlier in the development cycle, as well as improving the ability to find and fix bugs. In this article, I'll review the improvements added to Visual Studio 2015 for both .NET and C++ developers.

I'll begin by looking at two usability improvements and the addition of performance tools that work during debugging for both the Microsoft .NET Framework and C++. I'll then dig into a set of improvements specifically for .NET development and end with new developments for C++ developers.

Breakpoint Settings

Breakpoints are a fundamental capability of the debugger and chances are high that you used them the last time you fired up the debugger. There are times when using conditional breakpoints can help find the cause of a bug much faster. In Visual Studio 2015, it's easier to take advantage of conditional breakpoints by introducing a non-modal breakpoint settings window that sits in context with your code. It also enables you to easily combine different settings within the same window.

This article discusses:

- Debugging everyday problems with Visual Studio 2015.
- Diagnosing performance problems with Visual Studio 2015
- Debugging memory issues with Visual Studio 2015
- Being more productive with debugging tasks

Technologies discussed:

Microsoft .NET Framework, C++, Debugging, Performance Profiling, Edit and Continue, Memory

To review, Visual Studio 2015 offers the following settings for breakpoints:

- **Conditional expressions** break only when specified conditions are met. This is the logical equivalent of adding an if statement to the code and placing the breakpoint inside the if so it is only hit when the conditions are true. Conditional expressions are useful when you have code that's called multiple times but the bug only occurs with a specific input, so it would be tedious to have to manually check the values and then resume debugging.
- **Hit counts** break only after the breakpoint has been hit a certain number of times. These are useful in situations where code is called multiple times and you either know exactly when it's failing or have a general idea that "it fails after at least" a certain number of iterations.
- **Filters** break when the breakpoint is hit on a specific thread, process or machine. Filter breakpoints are useful for debugging code running in parallel when you only want to stop in a single instance.
- **Log a message** (called TracePoints) prints a message to the output window and is capable of automatically resuming execution. TracePoints are useful for doing temporary logging when you need to trace something and don't want to have to break and manually track each value.

To illustrate the value of conditional breakpoints, consider the example shown in **Figure 1**, where applications are retrieved by type inside a loop. The code is working for most input strings, failing only when the input is "desktop." One option is to set a breakpoint and then inspect the value of appType each time it's hit until it's "desktop," so I can begin stepping through the app to see what's not working. However, it'll be far quicker to create a breakpoint with a conditional expression that only breaks when appType is equal to "desktop," as shown in **Figure 1**.

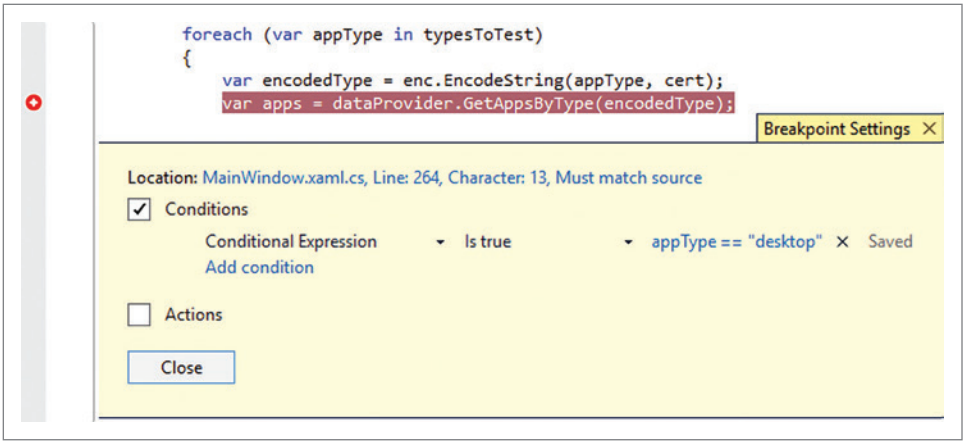


Figure 1 **Breakpoint Settings Window with a Conditional Expression**

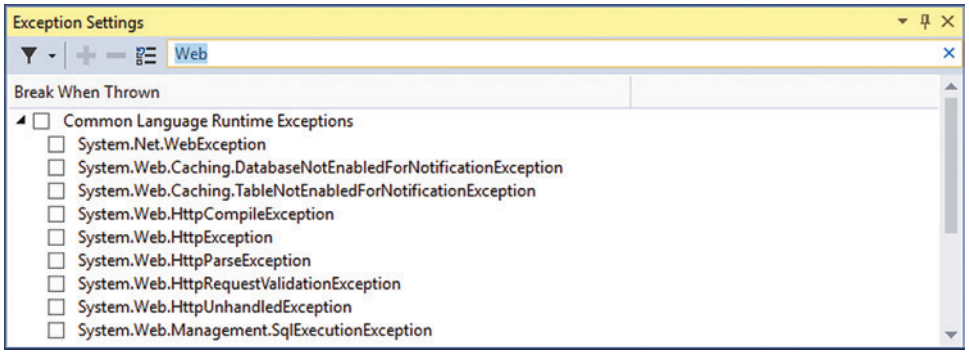


Figure 2 **The Visual Studio 2015 Exception Settings Window with a Search for All Exception Types That Contain "Web"**

By leveraging breakpoints with a conditional expression, it reduces the manual steps required to get the application into the correct state for debugging.

Exception Settings

As a developer, you know that exceptions can occur while your app is running. In many cases, you need to account for the possibility that something will go wrong by adding try/catch statements. For example, when an application retrieves information via a network call, that call can throw an exception if the user doesn't have a working network connection or if the server is unresponsive. In this case, the network request needs to be placed inside of a try, and if an exception occurs, the application should display an appropriate error message to the user. If the request fails when you expect it to work (because the URL is being incorrectly formatted in code, for example), you might be tempted to search through code to look for a place to set a breakpoint or to remove the try/catch so that the debugger will break on the unhandled exception.

The more efficient approach, however, is to configure the debugger to break when the exception is thrown using the Exception Settings dialog; this lets you set the debugger to break when all exceptions, or only exceptions of a certain type, are thrown. In previous versions of Visual Studio, feedback was that the Exception Settings dialog was too slow to open and had poor search functionality. So in Visual Studio 2015 the old Exception Settings dialog was replaced with a new modern Exception Settings window that opens instantly and offers the fast, consistent search feature that you've come to expect, as shown in **Figure 2**.

Performance Tools While Debugging

Your end users increasingly expect software to be fast and responsive, and exposing users to spinners or sluggish UIs can negatively affect user satisfaction and retention. Time is valuable and when users have a choice between applications with similar functionality, they'll pick the one that has the better UX.

However, when writing software, you often defer on proactive performance tuning and instead follow best practices, hoping the application will be fast enough. The main reason for that is it's time-consuming and difficult to measure performance and it can be even more difficult to find ways to improve it. To help with this, in Visual Studio 2015 the Visual Studio Diagnostics team integrated a set of performance tools directly into the debugger, called PerfTips and the Diagnostic Tools window. These tools help you learn about the performance of your code as a part of everyday debugging so that you can catch issues early and make informed design decisions that let you build your application for performance from the ground up.

An easy way to understand the performance of your application is to simply step through it using the debugger to get a feel for how long each line of code takes to execute. Unfortunately, this isn't very scientific as it depends on the ability to perceive differences. So it's hard to tell the difference between an operation that takes 25 ms versus an operation that takes 75 ms. Alternatively, you can modify the code to add timers that capture accurate information, but that comes at the inconvenience of changing code.

PerfTips solve this by showing you how long the code took to execute by placing the time in milliseconds on the right end of the

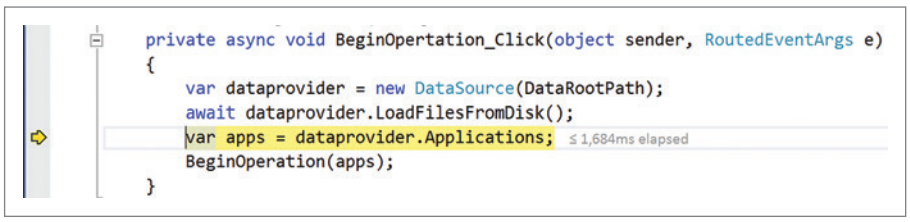


Figure 3 **PerfTip Showing the Elapsed Time It Took to Step over a Function Call**

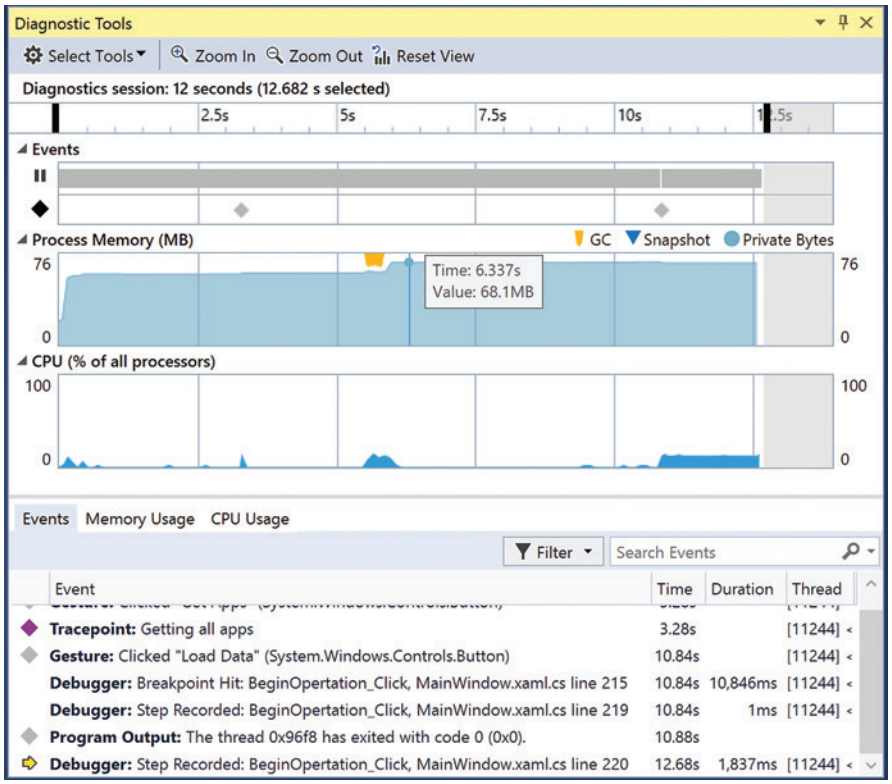


Figure 4 **Diagnostic Tools Window with the Events Tab Selected**

current line when the debugger stopped, as shown in **Figure 3**. PerfTips show the time it takes for the application to run between any two-break states in the debugger. This means they work both when stepping through code and running to breakpoints.

Let's look at a quick example of how a PerfTip can help you know how long code takes to execute. You have an application that loads files from the disk when the user clicks a button and then processes them accordingly. The expectation is that it'll take only a few milliseconds to load the files from the disk; however, using PerfTips, you can see it takes significantly longer. Based on this information, you can modify the application's design so it doesn't rely on loading all of the files when the user clicks the button early in the development cycle and before the change becomes too costly. To learn more about PerfTips, visit aka.ms/perftips.

The Diagnostic Tools window shows a history of the CPU and Memory usage of the application and all debugger break events (breakpoints, steps, exceptions and break all). The window offers three tabs: Events, Memory Usage and CPU Usage. The Events tab shows a history of all debugger break events, which means it contains a complete record of all PerfTips values. Additionally, with the Enterprise version of Visual Studio 2015, it contains all IntelliTrace events (covered later in this article). **Figure 4** shows the Events tab in Visual Studio 2015 Enterprise. Additionally, by having CPU and memory information update during your debugging session, it gives you the ability to see the CPU and memory characteristics of specific sections of code. For example, you can step over a method call and watch how the graphs change measuring the impact of that specific method.

The memory graph enables you to see the total memory usage of your application and spot trends in the application's memory use. For example, the graph might show a steady upward trend that would indicate the application is leaking memory and could eventually crash. I'll start by looking at how it works for the .NET Framework, and then cover the experience for C++.

When debugging the .NET Framework, the graph shows when Garbage Collections (GCs) occur, as well as total memory; this helps to spot situations where the overall memory use of the application is at an acceptable level but the performance of the application might be suffering due to frequent GCs (commonly caused by allocating too many short-lived objects). The Memory Usage tab lets you take snapshots of the objects in memory at any given point in time via the Take Snapshot button. It also provides the ability to compare two different snapshots, which is the easiest way to identify a memory leak. You take a snapshot, continue to use the application for a time period in a manner that you expect should be memory-neutral, and then take a second snapshot. **Figure 5** shows the memory tool with two .NET snapshots.

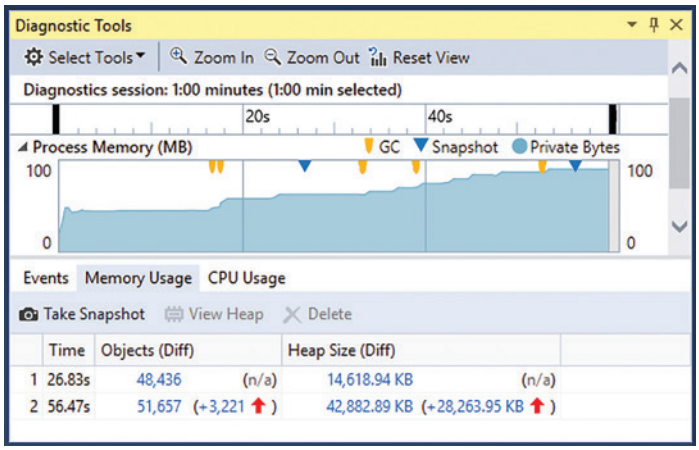


Figure 5 **Memory Usage Tab of the Diagnostic Tools Window with Two Snapshots**

When you click on a snapshot, a second window called the Heap view opens, which shows you the details of what objects are in memory, including the total number and their memory footprint. The bottom half of the Heap view shows what's holding references to the objects, preventing them from being garbage collected (referred to as Paths to Root), as well as what other types the selected type is referencing in the Referenced Types tab. **Figure 6** shows the Heap view with the differences between the two snapshots.

The C++ memory tool tracks memory allocations and deallocations to know what's in memory at any given point. To view the data, use the Take Snapshot button to create a record of the

Data entry errors. Consolidation headaches.
Quality data shouldn't be a pain.

Try a little Vitamin Q.

We supply Vitamin Q – "quality" – in data quality management solutions that profile, clean, enrich, and match your customer data – and keep it healthy over time. Add Vitamin Q to your data integration, business intelligence, Big Data, and CRM initiatives to ensure accurate data for greater insight and business value.

ACT NOW

Get a FREE 30-day trial!
www.MelissaData.com/vitaminQ

Solutions for 240+ Countries

10,000+ Customers Worldwide

30+ Years Strong

1985

Data Quality & Mailing Solutions

Cloud • On-Premise • Services

Germany
www.MelissaData.de

United Kingdom
www.MelissaData.co.uk

India
www.MelissaData.in

Australia
www.MelissaData.com.au

MELISSA DATA®
Your Partner in Global Data Quality
www.MelissaData.com | 1-800-MELISSA

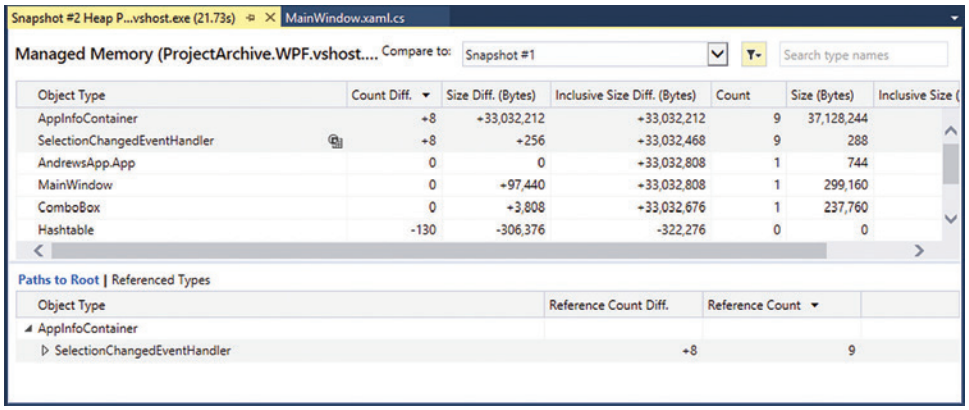


Figure 6 **Heap Snapshot View Shows the Differences Between Two .NET Snapshots**

allocation information at that moment in time. Snapshots also can be compared to see what memory has changed between two snapshots, making it much easier to track down memory leaks in code paths you expect to completely free up in memory. When you select a snapshot to view, the Heap view shows you a list of types along with their sizes, and in the case of comparing two snapshots, the differences between those numbers. When you see a type whose memory consumption you'd like to understand better, choose to view the instances of that type. The instances view shows you how big each instance is, how long it has been alive in memory and the call stack that allocated the memory. **Figure 7** shows the instances view.

The CPU graph shows CPU consumption of the application as a percentage of all cores on the machine. This means that it's useful for identifying operations that cause unnecessary spikes in CPU consumption, and can be useful for identifying operations that aren't taking full advantage of the CPU. Consider an example of processing a large amount of data where each record can be processed independently. When debugging the operation, you notice that the CPU graph on a machine with four cores is hovering slightly below 25 percent. This indicates that there's opportunity to parallelize the data processing across all of the cores on the machine to achieve faster performance of the application.

In Visual Studio 2015 Update 1, the Visual Studio Diagnostics team took it a step further and added a debugger-integrated CPU profiler into the CPU Usage tab that shows a breakdown of what functions in the application are using the CPU, as shown in **Figure 8**. For example, there's code that validates an e-mail address entered by the user is a valid format using a regular expression. When a valid e-mail address is entered, the code executes extremely fast; however, if an improperly formatted e-mail address is entered, the PerfTip shows that it can take close to two seconds for it to determine the address isn't valid. Looking at the Diagnostic Tools window, you see that there was a spike in CPU during that time. Then looking at the call tree in the CPU usage tab, you see that the regular expression matching is what's consuming the CPU, which is also shown in **Figure 8**. It turns out that C# regular expressions have a drawback: If they fail to match complex statements, the cost to process the entire string can be high. Using PerfTips and the CPU usage tool in the Diagnostic Tools window you're quickly able to determine the performance of the application will not be acceptable in all cases using the regular expression. So you can modify your code to use some standard string operations instead that yield much better performance results when bad data is input. This could have been a costly bug to fix later, especially if it made it all the way to production. Fortunately, the debugger-integrated tools enabled the design to change during development to ensure consistent performance. To learn more about the Diagnostic Tools window go to aka.ms/diagtoolswindow.

Next, let's look at some of the improvements made that are specifically for .NET debugging.

Lambda Support in the Watch and Immediate Windows

Lambda expressions (such as LINQ) are an incredibly powerful and common way to quickly deal with collections of data. They

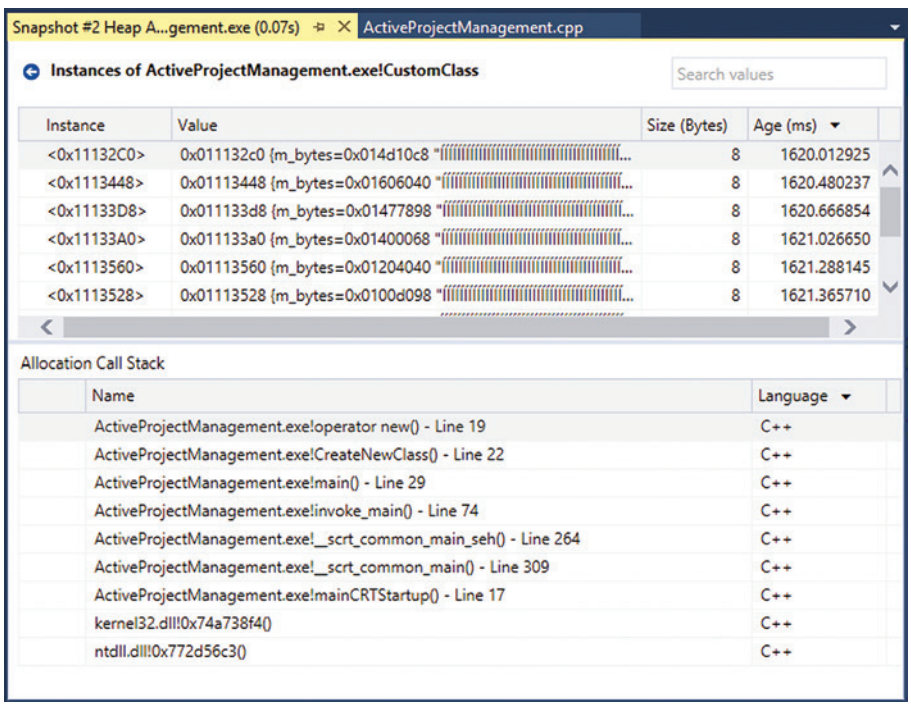


Figure 7 **Heap View of C++ Memory Showing the Instances View with the Allocating Call Stack**

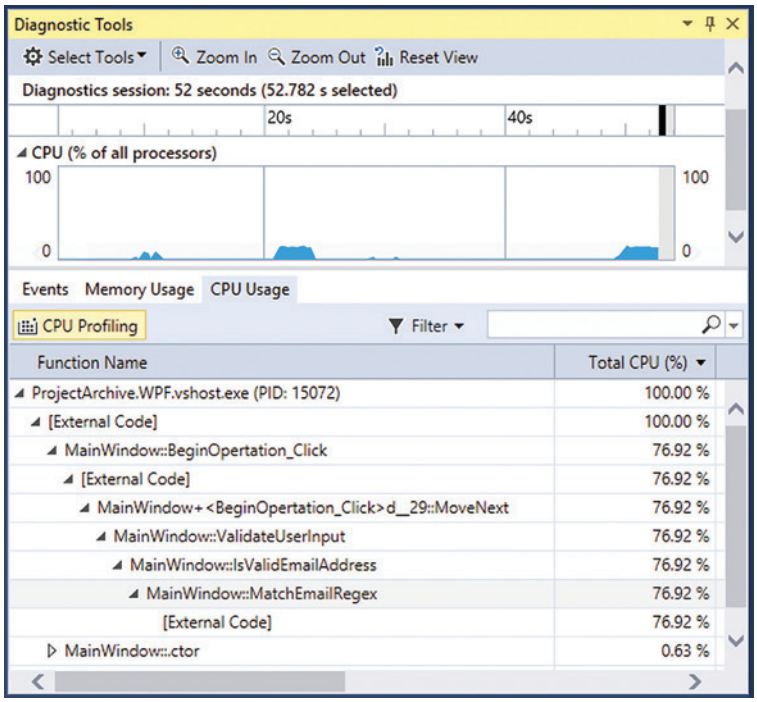


Figure 8 **CPU Usage Tab Showing the CPU Consumption of Matching a Regular Expression**

enable you to do complex operations with a single line of code. Often, you find yourself wanting to test changes to expressions in the debugger windows, or use LINQ to query a collection rather than manually expanding it in the debugger. As an example, consider a situation where your application is querying a collection of items and returning zero results. You're sure there are items that match the intended criteria, so you start by querying the collection to extract the distinct elements in the list. The results confirm that there are elements that match your intended criteria, but there appears to be a mismatch with string casing; still, you don't care if the case matches exactly. Your hypothesis is that you need to modify the query to ignore casing when it does the string comparison. The easiest way to test that hypothesis is to type the new query into the Watch or Immediate window and see if it returns the expected results, as shown in **Figure 9**.

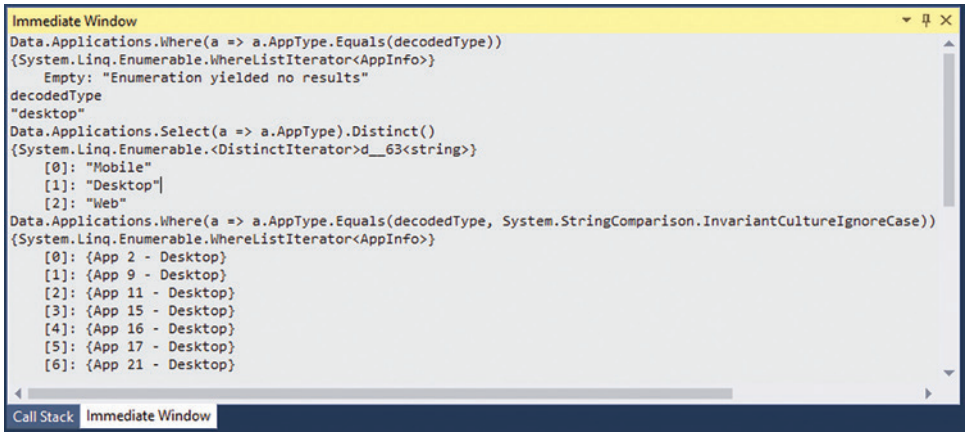


Figure 9 **Immediate Window with Two Evaluated Lambda Expressions**

Unfortunately, in versions of Visual Studio prior to 2015, typing a Lambda expression into a debugger window would result in an error message. Therefore, to address this top feature request, support was added for using Lambda expressions in the debugger windows.

.NET Edit and Continue Improvements

A favorite debugging productivity feature in Visual Studio is Edit and Continue. Edit and Continue lets you change code while stopped in the debugger, then have the edit applied without the need to stop debugging, recompile and run the application to the same location to verify the change fixed the problem. However, one of the most frustrating things when using Edit and Continue is to make the edit, attempt to resume execution and see a message that the edit you made couldn't be applied while debugging. This was becoming a more common problem as the framework continued to add new language features that Edit and Continue could not support (Lambda expressions and async methods, for example).

To improve this, Microsoft added support for several previously unsupported edit types that will significantly increase the number of times edits can be successfully applied during debugging. Improvements include the ability to modify Lambda expressions, edit anonymous and async methods, work with dynamic types and modify C# 6.0 features (such as string interpolation and null conditional operators). For a complete list of supported edits, visit aka.ms/dotnetenc. If you make an edit and receive an error message that the edit cannot be applied, make sure to check the Error List as the compiler places additional information about why the edit could not be compiled using Edit and Continue.

Additional improvements for Edit and Continue include support for applications using x86 and x64 CoreCLR, meaning it can be used when debugging Windows Phone apps on the emulator, and support for remote debugging.

New IntelliTrace Experience

IntelliTrace provides historical information about your application to help take the guesswork out of debugging in the Enterprise edition and get you to the relevant parts of your code faster, with less debug sessions. A comprehensive set of improvements were added to IntelliTrace to make it easier than ever to use. Improvements include a timeline that shows when in time events occur, the ability to see events in real time, support for TracePoints and integration into the Diagnostic Tools window.

The timeline enables you to understand when in time events occur and spot groups of events, which might be related. Events appear live in the Events tab, where in prior versions you needed to

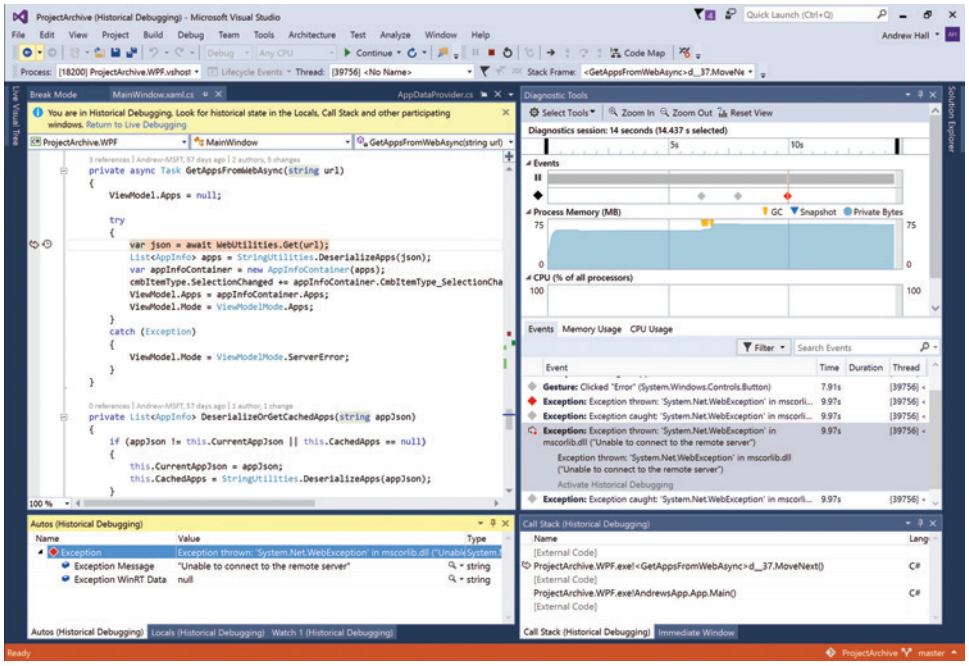


Figure 10 **Visual Studio in Historical Debugging Mode at the Location an Exception Was Thrown**

enter a break state in the debugger to see the events IntelliTrace collected. TracePoint integration lets you create custom IntelliTrace events using standard debugger functionality. Finally, Diagnostic Tools window integration puts IntelliTrace events in context with performance information, letting you use the rich information of IntelliTrace to understand the cause of performance and memory issues by correlating the information across a common timeline.

When you see a problem in the application, you normally form a hypothesis about where to start investigating, place a breakpoint and run the scenario again. If the problem turns out not to be in that location, you need to form a new hypothesis about how to get to the correct location in the debugger and start the process again. IntelliTrace aims to improve this workflow by removing the need to re-run the scenario again.

Consider the example from earlier in this article where a network call is failing unexpectedly. Without IntelliTrace, you need to see the failure the first time, enable the debugger to break when the exception is thrown, then run the scenario again. With IntelliTrace, when you see the failure, all you have to do is look in the Events tab of the Diagnostic Tools window. The exception appears as an event; select it and click Activate Historical Debugging. You're then navigated back in time to the location in the source where the exception occurred, the Locals and Autos windows show the exception information and the Call Stack window is populated with the call stack where the exception occurred, as shown in **Figure 10**.

Finally, let's look at some of the most notable improvements that were made for C++ debugging in Visual Studio 2015.

C++ Edit and Continue

As mentioned in earlier, Edit and Continue is a productivity feature that enables you to modify your code while stopped in the debugger and then have the edits applied when you resume execution without the need to stop debugging to recompile the modified application. In previous versions, C++ Edit and Continue had two notable limitations. First, it only supported x86 applications. Second, enabling Edit and Continue resulted in Visual Studio using the Visual Studio 2010 C++ debugger, which lacked new functionality such as support for Natvis data visualizations (see aka.ms/natvis). In Visual Studio 2015, both of these gaps were filled. Edit and Continue is enabled by default for C++ projects for both x86 and x64 applications and it can even work when attaching to processes and remote debugging.

Android and iOS Support

As the world moves to a mobile-first mentality, many organizations are finding the need to create mobile applications. With the diversification of platforms, C++ is one of the few technologies that can be used across any device and OS. Many large organizations are using shared C++ code for common business logic they want to re-use across a broad spectrum of offerings. To facilitate this, Visual Studio 2015 offers tools that enable mobile developers to target Android and iOS directly from Visual Studio. This includes the familiar Visual Studio debugging experience developers use for daily work in C++ on Windows and the Visual Studio Emulator for Android, a fast, free Android Emulator available both when you install Visual Studio 2015 and as a standalone tool (to learn more about the Emulator, visit aka.ms/vsemu).

Wrapping It Up

The Visual Studio Diagnostics team is extremely excited about the broad range of functionality that's been added to the debugging experience in Visual Studio 2015. Everything covered in this article with the exception of IntelliTrace is available in the Community edition of Visual Studio 2015.

You can continue to follow the team's progress and future improvements on the team blog at aka.ms/diagnosticsblog. Please try out the features discussed here and give the team feedback on how it can continue to improve Visual Studio to meet your debugging needs. You can leave comments and questions on the blog posts, or send feedback directly through Visual Studio by going to the Send Feedback icon in the top-right part of the IDE. ■

Andrew Hall *is the program manager lead for the Visual Studio Diagnostics team that builds the core debugging, profiling and IntelliTrace experiences, as well as the Android Emulator for Visual Studio. Over the years, he has directly worked on the debugger, profiler and code analysis tools in Visual Studio.*

Thanks to the following technical experts for reviewing this article: *Angelos Petropoulos, Dan Taylor, Kasey Uhlenhuth and Adam Welch*

BEST FILE APIs

Open Create Convert Print Save

files from your *applications*!

Contact Us:
US: +1 888 277 6734
EU: +44 141 416 1112
AU: +61 2 8003 5926
sales@aspose.com

SCAN FOR
20% SAVINGS!

BUSINESS FILE FORMATS

ASPOSE.Cells

XLS, CSV, PDF, SVG, HTML, PNG
BMP, XPS, JPG, SpreadsheetML...

ASPOSE.Words

DOC, RTF, PDF, HTML, PNG
ePUB, XML, XPS, JPG...

ASPOSE.Pdf

PDF, XML, XSL-FO, HTML, BMP
JPG, PNG, ePUB...

ASPOSE.Slides

PPT, POT, POTX, XPS, HTML
PNG, PDF...

ASPOSE.BarCode

JPG, PNG, BMP, GIF, TIF, WMF
ICON...

ASPOSE.Tasks

XML, MPP, SVG, PDF, TIFF
PNG...

ASPOSE.Email

MSG, EML, PST, EMLX
OST, OFT...

ASPOSE.Imaging

PDF, BMP, JPG, GIF, TIFF
PNG...

+ MANY MORE!

Get your FREE evaluation copy at www.aspose.com

.NET | Java | Cloud

ASPOSE
Your File Format APIs

Managed Profile-Guided Optimization Using Background JIT

Hadi Brais

Some performance optimizations performed by a compiler are always good. That is, no matter which code actually gets executed at run time, the optimization will improve performance. Consider, for example, loop unrolling to enable vectorization. This optimization transforms a loop so that instead of performing a single operation in the body of the loop on a single set of operands (such as adding two integers stored in different arrays), the same operation would be performed on multiple sets of operands simultaneously (adding four pairs of integers).

On the other hand, there are extremely important optimizations that the compiler performs heuristically. That is, the compiler doesn't know for sure that these optimizations will actually work great for the code that gets executed at run time. The two most important optimizations that fall in this category (or probably among all categories) are register allocation and function inlining. You can help the compiler make better decisions when performing such optimizations by running the app one or more times and providing it with typical user input while at the same time recording which code got executed.

The information that has been collected about the execution of the app is called a profile. The compiler can then use this profile to make some of its optimizations more effective, sometimes resulting in significant speedups. This technique is called profile-guided optimization (PGO). You should use this technique when you've written readable, maintainable code, employed good algorithms, maximized data access locality, minimized contention on locks and turned on all possible compiler optimizations, but still aren't satisfied with the resulting performance. Generally speaking, PGO can be used to improve other characteristics of your code, not just performance. However, the technique discussed in this article can be used to only improve performance.

I've discussed in detail native PGO in the Microsoft Visual C++ compiler in a previous article at msdn.com/magazine/mt422584. For those who read that article, I've got some great news. Using managed PGO is simpler. In particular, the feature I'm going to discuss in this article, namely background JIT (also called multicore JIT) is a lot simpler. However, this is an advanced article. The CLR team wrote an introductory blog post three years ago (bit.ly/1Znlj9y).

This article discusses:

- The overhead introduced when invoking the JIT compiler to compile IL code
- Using background JIT to improve startup performance of common usage scenarios
- Details of how background JIT hides the JIT compilation overhead in a background thread
- When to use background JIT
- Using PerfView to evaluate the gained benefit of background JIT

Technologies discussed:

Microsoft .NET Framework, JIT, C#, PerfView

Background JIT is supported in the Microsoft .NET Framework 4.5 and all later versions.

There are three managed PGO techniques:

- Compile managed code to binary code using Ngen.exe (a process known as preJIT) and then use Mpgo.exe to generate profiles representing common usage scenarios that can be used to optimize the performance of the binary code. This is similar to native PGO. I'll refer to this technique as static MPGO.
- The first time an Intermediate Language (IL) method is about to be JIT compiled, generate instrumented binary code that records information at run time regarding which parts of the method are getting executed. Then later use that in-memory profile to re-JIT compile the IL method to generate highly optimized binary code. This is also similar to native PGO except that everything happens at run time. I'll refer to this technique as dynamic MPGO.
- Use background JIT to hide as much as possible the overhead to JIT by intelligently JIT compiling IL methods before they actually get executed for the first time. Ideally, by the time a method is called for the first time, it would've already been JIT compiled and there would be no need to wait for the JIT compiler to compile the method.

Interestingly, all of these techniques have been introduced in the .NET Framework 4.5, and later versions also support them. Static MPGO works only with native images generated by Ngen.exe. In contrast, dynamic MPGO works only with IL methods. Whenever possible, use Ngen.exe to generate native images and optimize them using static MPGO because this technique is much simpler, while at the same time it gives respectable speedups. The third technique, background JIT, is very different from the first two because it reduces JIT compiling overhead rather than improving the performance of the generated binary code and, therefore, can be used together with either of the other two techniques. However, using background JIT alone can sometimes be very beneficial and improve the performance of the app startup or a particular common usage scenario by up to 50 percent, which is great. This article focuses exclusively on background JIT. In the next section, I'll discuss the traditional way of JIT compiling IL methods and how it impacts performance. Then I'll discuss how background JIT works, why it works that way and how to properly use it.

Traditional JIT

You probably already have a basic idea of how the .NET JIT compiler works because there are many articles that discuss this process. However, I would like to revisit this subject in a little more detail and accuracy (but not much) before I get to background JIT so that you can easily follow the next section and understand the feature firmly.

Consider the example shown in **Figure 1**. T0 is the main thread. The green parts of the thread indicate that the thread is executing app code and it's running at full speed. Let's assume that T0 is executing in a method that has already been JIT compiled (the topmost green part) and the next instruction is to call the IL method M0. Because this is the first time M0 will get executed and because it's represented in IL, it has to be compiled to binary code that the processor can execute. For this reason, when the call instruction is executed, a function known as the JIT IL stub is called. This function eventually calls the JIT compiler to JIT the IL code of M0 and returns the address of the generated binary code. This work has nothing to do with the app itself and is represented by the red part of T0 to indicate that it's an overhead. Fortunately, the memory location that stores the address of the JIT IL stub will be patched with the address of the corresponding binary code so that future calls to the same function run at full speed.

Now, after returning from M0, some other code that has already been JIT compiled is executed and then the IL method M1 is called. Just like with M0, the JIT IL stub is called, which in turn calls the JIT compiler to compile the method and returns the address of the binary code. After returning from M1, some more binary code is executed and then two more threads, T1 and T2, start running. This is where things get interesting.

After executing methods that have already been JIT compiled, T1 and T2 are going to call the IL method M3, which has never been called before and, therefore, has to be JIT compiled. Internally, the JIT compiler maintains a list of all methods that are being JIT compiled. There's a list for each AppDomain and one for shared code. This list is protected by a lock and every element is also protected by its own lock so that multiple threads can safely engage in JIT compilation simultaneously. What will happen in this case is that one thread, say T1, will be JIT compiling the method and wasting time doing work that has nothing to do with the app while T2 is doing nothing—waiting on a lock simply because it actually has nothing to do—until the binary code of M3 becomes available. At the same time, T0 will be compiling M2. When a thread finishes JIT compiling a method, it replaces the address of the JIT IL stub with the address of the binary code, releases the locks and executes the method. Note that T2 will eventually wake up and just execute M3.

The rest of the code that's executed by these threads is shown in the green bars in **Figure 1**. This means that the app is running

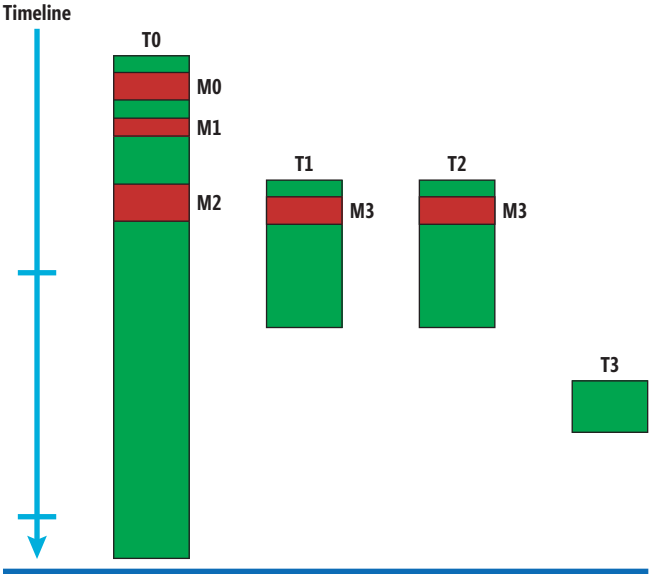


Figure 1 **The Overhead of Traditional JIT When Executing Managed Code**

at full speed. Even when a new thread, T3, starts running, all the methods that it needs to execute have already been JIT compiled and, therefore, it also runs at full speed. The resulting performance becomes very close to native code performance.

Roughly speaking, the duration of each of these red segments mainly depends on the amount of time it takes to JIT the method, which in turn depends on how large and complex the method is. It can range from a few microseconds to tens of milliseconds (excluding the time to load any required assemblies or modules). If the startup of an app requires executing for the first time less than a hundred methods, it's not a big deal. But if it requires executing for the first time hundreds or thousands of methods, the impact of all of the resulting red segments might be significant, especially when the time it takes to JIT a method is comparable to the time it takes to execute the method, which causes a double-digit percentage slowdown. For example, if an app requires executing a thousand different methods at startup with an average JIT time of 3 milliseconds, it would take 3 seconds to complete startup. That's a big deal. It's not good for business because your customers will not be satisfied.

Note that there is a possibility more than one thread JIT compiles the same method at the same time. It's also possible that the first attempt to JIT fails but the second succeeds. Finally, it's also possible that a method that has already been JIT compiled is re-JIT compiled. However, all of these cases are beyond the scope of this article and you don't have to be aware of them when using background JIT.

Background JIT

The JIT compiling overhead discussed in the previous section cannot be avoided or significantly reduced. You have to JIT IL methods to execute them. What you can do, however, is change the time at which this overhead is incurred. The key insight is that instead of waiting for an IL method to be called for the first time to JIT it, you can JIT that method earlier so that by the time it's called, the binary code would have already been generated. If you got it right, all threads shown in **Figure 1** would be green and they would all run at full speed, as if you're executing an NGEN native image or maybe better. But before you get there, two problems must be addressed.

The first problem is that if you're going to JIT a method before it's needed, which thread is going to JIT it? It's not hard to see that the best way to solve this problem is to have a dedicated thread that runs in the background and JIT methods that are likely to be executed as quickly as possible. As a consequence, this only works if at least two cores are available (which is almost always the case) so that the JIT compiling overhead is hidden by the overlapping execution of the app's code.

The second problem is this: How do you know which method to JIT next before it's called for the first time? Keep in mind that typically there are conditional method calls in every method and so you can't just JIT all methods that might be called or be too speculative in choosing which methods to JIT next. It's very likely that the JIT background thread falls behind the app threads very quickly. This is where profiles come into play. You first exercise the startup of the app and any common usage scenarios and record which methods were JIT compiled and the order in which they were JIT compiled for each scenario separately. Then you can publish the app together with the recorded profiles so that when it runs on the user's machine, the JIT compiling overhead will be minimized with respect to the wall clock time (this is how the user perceives time and performance). This feature is called background JIT and you can use it with very little effort from your side.

> If an app requires executing a thousand different methods at startup with an average JIT time of 3 milliseconds, it would take 3 seconds to complete startup. That's a big deal.

In the previous section, you saw how the JIT compiler can JIT compile different methods in parallel in different threads. So technically, that traditional JIT is already multicore. It's unfortunate and confusing that the MSDN documentation refers to the feature as multicore JIT, based on the requirement of at least two cores rather than based on its defining characteristic. I'm using the name "background JIT" because this is the one I'd like to spread. PerfView has built-in support for this feature and it uses the name background JIT. Note that the name "multicore JIT" is the name used by Microsoft early in development. In the rest of this section, I'll discuss all you have to do to apply this technique on your code and how it changes the traditional JIT model. I'll also show you how to use PerfView to measure the benefit of background JIT when you use it on your own apps.

To use background JIT, you need to tell the runtime where to put the profiles (one for each scenario that triggers a lot of JIT compilation). You also need to tell the runtime which profile to use so that it reads the profile to determine which methods to compile on the background thread. This, of course, has to be done sufficiently before the associated usage scenario starts.

To specify where to put the profiles, call the System.Runtime.ProfileOptimization.SetProfileRoot method defined in mscorlib.dll. This method looks like this:

```
public static void SetProfileRoot(string directoryPath);
```

The purpose of its only parameter, directoryPath, is to specify the directory of the folder in which all profiles will be read from or written to. Only the first call to this method in the same AppDomain takes effect and any other calls are ignored (however, the same path can be used by different AppDomains). Also, if the computer doesn't have at least two cores, any call to SetProfileRoot is ignored. The only thing that this method does is store the specified directory in an internal variable so that it can be used whenever required later. This method is usually called by the

executable (.EXE) of the process during initialization. Shared libraries should not call it. You can call this method any time while the app is running, but before any call to the ProfileOptimization.StartProfile method. This other method looks like this:

```
public static void StartProfile(string profile);
```

When the app is about to go through an execution path whose performance you would like to optimize (such as startup), call this method and pass to it the file name and extension of the profile. If the file doesn't exist, a profile is recorded and stored in a file with the specified name in the folder you've specified using SetProfileRoot. This process is called "profile recording." If the specified file exists and contains a valid background JIT profile, background JIT takes effect in a dedicated background thread JIT compiling methods, chosen according to what the profile says. This process is called "profile playing." While playing the profile, the behavior exhibited by the app will still be recorded and the same input profile will be replaced.

You can't play a profile without recording; it's just not currently supported. You can call StartProfile multiple times specifying different profiles suitable for different execution paths. This method has no effect if it has been called before initializing the profile root using SetProfileRoot. Also, both methods have no effect if the specified argument is invalid in any way. In fact, these methods don't throw any exceptions or return error codes to not impact the behavior of apps in any undesirable way. Both of them are thread-safe, just like every other static method in the framework.

> To use background JIT, you need to tell the runtime where to put the profiles (one for each scenario that triggers a lot of JIT compilation). You also need to tell the runtime which profile to use so that it reads the profile to determine which methods to compile on the background thread.

For example, if you want to improve startup performance, call these two methods as a first step in the main function. If you want to improve the performance of a particular usage scenario, call StartProfile when the user is expected to initiate that scenario and call SetProfileRoot anytime earlier. Remember that everything happens locally in AppDomains.

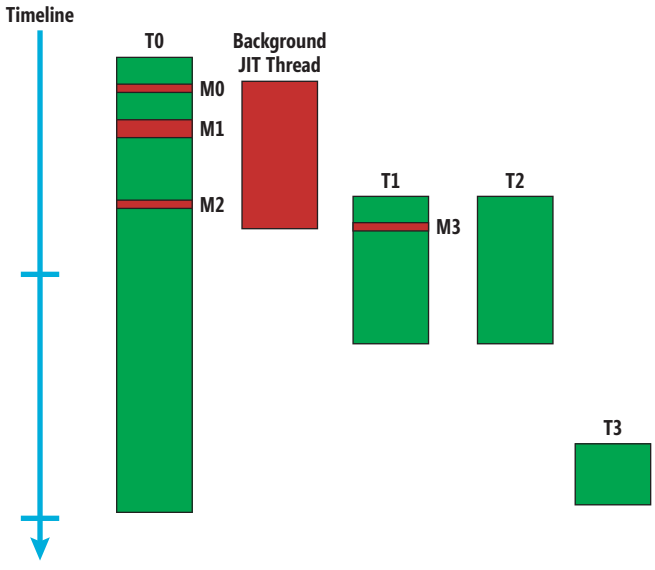


Figure 2 **An Example Showing the Background JIT Optimization as Compared to Figure 1**

That's all you have to do to use background JIT in your code. It's so simple that you can just try it without thinking too much about whether it will be useful or not. You can then measure the gained speedup to determine whether it's worth keeping. If the speedup is at least 15 percent, you should keep it. Otherwise, it's your call. Now I'll explain in detail how it works.

Every time StartProfile is called, the following actions are performed in the context of the AppDomain in which the code is currently executing:

1. All the contents of the file that contains the profile (if it exists) are copied into memory. The file is then closed.
2. If this isn't the first time StartProfile has been successfully called, there would already be a background JIT thread running. In this case, it's terminated and a new background thread is created. Then the thread that called StartProfile returns to the caller.
3. This step happens in the background JIT thread. The profile is parsed. The recorded methods are JIT compiled in the order they were recorded sequentially and as fast as possible. This step constitutes the profile playing process.

That's it as far as the background thread is concerned. If it has finished JIT compiling all the recorded methods, it will terminate silently. If anything goes wrong while parsing or JIT compiling the methods, the thread terminates silently. If an assembly or module that hasn't been loaded and is required to JIT a method, it will not be loaded and so the method will not be JIT-compiled. Background JIT has been designed so that it doesn't change the behavior of the program as much as possible. When a module is loaded, its constructor is executed. Also, when a module can't be found, callbacks registered with the System.Reflection.Assembly.ModuleResolve event are called. Therefore, if the background thread loads a module earlier than it otherwise would be, the behavior of these functions might change. This similarly applies to callbacks registered with the System.AppDomain.AssemblyLoad event.

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

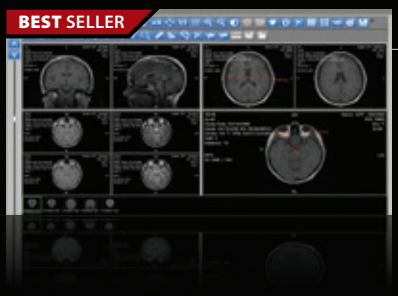
Aspose.Total for .NET | from $2,449.02

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 Medical Imaging SDKs V19 | from $4,995.00 SRP

LEAD TECHNOLOGIES

Powerful DICOM, PACS and HL7 functionality.

- Load, save, edit, annotate and display DICOM Data Sets with support for 2014 specifications
- High-level PACS Client and Server components and frameworks
- OEM-ready HTML5 Zero-footprint Viewer and DICOM Storage Server apps with source code
- Medical-specific image processing functions for enhancing 16-bit grayscale images
- Native libraries for .NET, C/C++, HTML5, JavaScript, WinRT, iOS, OS X, Android, Linux & more

BEST SELLER

DevExpress DXperience 15.2 | from $1,439.99

DevExpress

The complete range of DevExpress .NET controls and libraries for all major Microsoft platforms.

- WinForms Grid: New data-aware Tile View
- WinForms Grid & TreeList: New Excel-inspired Conditional Formatting
- .NET Spreadsheet: Grouping and Outline support
- ASP.NET: New Rich Text Editor-Word Processing control
- ASP.NET Reporting: New Web Report Designer

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

VISA DISCOVER

GSA Schedule
Contract GS-35F-0188R

Because the background JIT doesn't load the modules it needs, it might not be able to compile many of the recorded methods, leading to modest benefit.

You might be wondering, why not create more than one background thread to JIT more methods? Well, first, these threads are compute-intensive and so they might compete with app threads. Second, more of these threads mean more thread synchronization contention. Third, it's not unlikely that methods get JIT compiled but never get called by any app thread. Conversely, a method might get called for the first time that's not even recorded in the profile or before it gets JIT compiled by the multicore thread. Due to these issues, having more than one background thread might not be very beneficial. However, the CLR team might do this in the future (especially when the restriction of loading modules can be relaxed). Now it's time to discuss what happens in the app threads including the profile recording process.

Figure 2 shows the same example as in **Figure 1** except that background is JIT-enabled. That is, there is a background thread JIT compiling the methods M0, M1, M3 and M2, in that order. Notice how this background thread is racing against the app threads T0, T1, T2 and T3. The background thread has to JIT every method before it's called for the first time by any thread to fulfill its purpose in life. The following discussion assumes that this is the case with M0, M1 and M3, but not quite with M2.

When T0 is about to call M0, the background JIT thread has already JIT compiled it. However, the method address hasn't been patched yet and still points to the JIT IL stub. The background JIT thread could've patched it, but it doesn't in order to determine later whether the method has been called or not. This information is used by the CLR team to evaluate background JIT. So the JIT IL stub gets called and it sees that the method has already been compiled on the background thread. The only thing it has to do is patch the address and execute the method. Notice how the JIT compiling overhead has been completely eliminated on this thread. M1 receives the same treatment when called on T0. M3 receives the same treatment, as well, when called on T1. But, when T2 calls M3 (refer to **Figure 1**), the method address has been patched by T1 very quickly and so it directly calls the actual binary code of the method. Then T0 calls M2. However, the background JIT thread hasn't finished yet JIT compiling the method and, therefore, T0 waits on the JIT lock of the method. When the method is JIT compiled, T0 wakes up and calls it.

I haven't yet discussed how methods get recorded in the profile. It's also quite possible that an app thread calls a method that the background JIT thread hasn't even begun JIT compiling (or will never JIT it because it's not in the profile). I've compiled the steps performed on an app thread when it calls a static or a dynamic IL method that hasn't been JIT compiled yet in the following algorithm:

1. Acquire the JIT list lock of the AppDomain in which the method exists.
2. If the binary code has already been generated by some other app thread, release the JIT list lock and go to step 13.
3. Add a new element to the list representing the JIT worker of the method if it doesn't exist. If it already exists, its reference count is incremented.
4. Release the JIT list lock.
5. Acquire the JIT lock of the method.
6. If the binary code has already been generated by some other app thread, go to step 11.
7. If the method isn't supported by the background JIT, skip this step. Currently, background JIT supports only statically emitted IL methods that are defined in assemblies that haven't been loaded with System.Reflection.Assembly.Load. Now if the method is supported, check whether it has already been JIT compiled by the background JIT thread. If this is the case, record the method and go to step 9. Otherwise, go to the next step.
8. JIT the method. The JIT compiler examines the IL of the method, determines all required types, makes sure that all required assemblies are loaded and all required type objects are created. If anything goes wrong, an exception is thrown. This step incurs most of the overhead.
9. Replace the address of the JIT IL stub with the address of the actual binary code of the method.
10. If the method has been JIT compiled by an app thread rather than by the background JIT thread, there is an active background JIT recorder and the method is supported by background JIT; the method is recorded in an in-memory profile. The order in which methods were JIT compiled is maintained in the profile. Note that the generated binary code is not recorded.
11. Release the method JIT lock.
12. Safely decrease the reference count of the method using the list lock. If it becomes zero, the element is removed.
13. Execute the method.

The background JIT recording process terminates when any of the following situations occur:

- The AppDomain associated with the background JIT manager is unloaded for any reason.
- StartProfile is called again in the same AppDomain.
- The rate at which methods are JIT compiled in app threads becomes very small. This indicates that the app has reached a stable state where it rarely requires JIT compiling. Any methods that get JIT compiled after this point are not of interest to background JIT.
- One of the recording limits has been reached. The maximum number of modules is 512, the maximum number of methods is 16,384 and the longest continuous duration of recording is one minute.

When the recording process terminates, the recorded in-memory profile is dumped to the specified file. In this way, the next time the app runs, it picks up the profile that reflects the behavior exhibited by the app during its last run. As I've mentioned before, profiles are always overwritten. If you want to retain the current profile, you must manually make a copy of it before calling StartProfile. The size of a profile typically doesn't exceed a few dozen kilobytes.

Before closing this section, I'd like to talk about selecting profile roots. For client apps, you can either specify a user-specific directory or an app-relative directory, depending on whether you want to have different sets of profiles for different users or just one set of

profiles for all users. For ASP.NET and Silverlight apps, you'll probably be using an app-relative directory. In fact, starting with ASP.NET 4.5 and Silverlight 4.5, background JIT is enabled by default and the profiles are stored next to the app. The runtime will behave as if you've called SetProfileRoot and StartProfile in the main method and so you don't have to do anything to use the feature. You can still call StartProfile, though, as described earlier. You can turn off automatic background JIT by setting the profileGuidedOptimizations flag to None in the Web configuration file as described in the .NET Blog post, "An Easy Solution for Improving App Launch Performance" (bit.ly/1Znlj9y). This flag can take only one other value, namely All, which enables background JIT (the default).

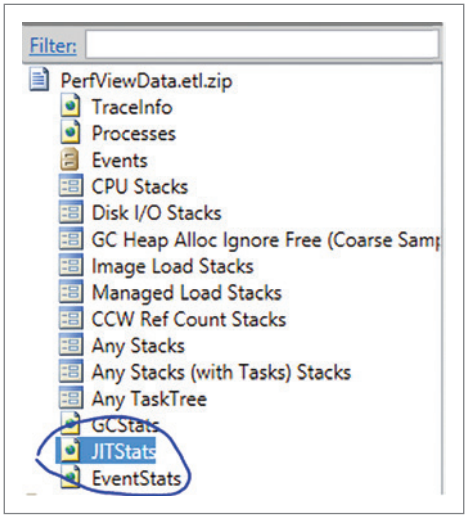


Figure 3 **The Location of JITStats in PerfView**

Background JIT in Action

Background JIT is an Event Tracing for Windows (ETW) provider. That is, it reports a number of events that are related to this feature to ETW consumers such as the Windows Performance Recorder and PerfView. These events enable you to diagnose any inefficiencies or failures that occurred in background JIT. In particular, you can determine how many methods were compiled on the background thread and the total JIT time of those methods. You can download PerfView from bit.ly/1PpJUpv (no installation required, just unzip and run). I'll use the following simple code for demonstration:

```
class Program {
  const int OneSecond = 1000;
  static void PrintHelloWorld() {
    Console.WriteLine("Hello, World!");
  }
  static void Main() {
    ProfileOptimization.SetProfileRoot(@"C:\Users\Hadi\Desktop");
    ProfileOptimization.StartProfile("HelloWorld Profile");
    Thread.Sleep(OneSecond);
    PrintHelloWorld();
  }
}
```

So now that you've generated a background JIT profile, use PerfView to launch and profile the executable.

In the main function, SetProfileRoot and StartProfile are called to set up background JIT. The thread is put to sleep for about one second and then a method, PrintHelloWorld, is called. This method simply calls Console.WriteLine and returns. Compile this code to an IL executable. Note that Console.WriteLined doesn't require JIT compiling because it's already been compiled using NGEN while installing the .NET Framework on your computer.

Use PerfView to launch and profile the executable (for more information on how to do that, refer to the .NET Blog post, "Improving Your App's Performance with PerfView," at bit.ly/1nablYC, or the Channel 9 PerfView Tutorial at bit.ly/23fwp6r). Remember to check the Background JIT checkbox (required only in .NET Framework 4.5 and 4.5.1) to enable capturing events from this feature. Wait until PerfView finishes and then open the JITStats page (see **Figure 3**); PerfView will tell you that the process doesn't use background JIT compilation. That's because in the first run, a profile has to be generated.

So now that you've generated a background JIT profile, use PerfView to launch and profile the executable. This time, however, when you open the JITStats page, you'll see that one method, namely PrintHelloWorld, was JIT compiled on the background JIT thread and one method, namely Main, wasn't. It'll also tell you that about 92 percent of JIT time that was spent compiling all IL methods occurred in app threads. The PerfView report will also show a list of all methods that were JIT compiled, the IL and binary size of each method, who JIT compiled the method and other information. You can also easily access the full set of information about the background JIT events. However, due to the lack of space here, I won't go into the details.

You might be wondering about the purpose of sleeping for about one second. This is necessary to have PrintHelloWorld JIT compiled on the background thread. Otherwise, it's likely that the app thread will start compiling the method before the background thread. In other words, you have to call StartProfile early enough so that the background thread can stay ahead most of the time.

Wrapping Up

Background JIT is a profile-guided optimization supported in the .NET Framework 4.5 and later. This article discussed almost everything you need to know about this feature. I've demonstrated why this optimization is needed, how it works and how to properly use it in your code in great detail. Use this feature when NGEN isn't convenient or possible. Because it's easy to use, you can just try it without thinking too much about whether it would benefit your app or not. If you're happy with the gained speedup, keep it. Otherwise, you can easily remove it. Microsoft used background JIT to improve the startup performance of some of its apps. I hope that you can effectively use it in your apps, as well, to achieve significant startup speedups of JIT-extensive usage scenarios and app startup. ■

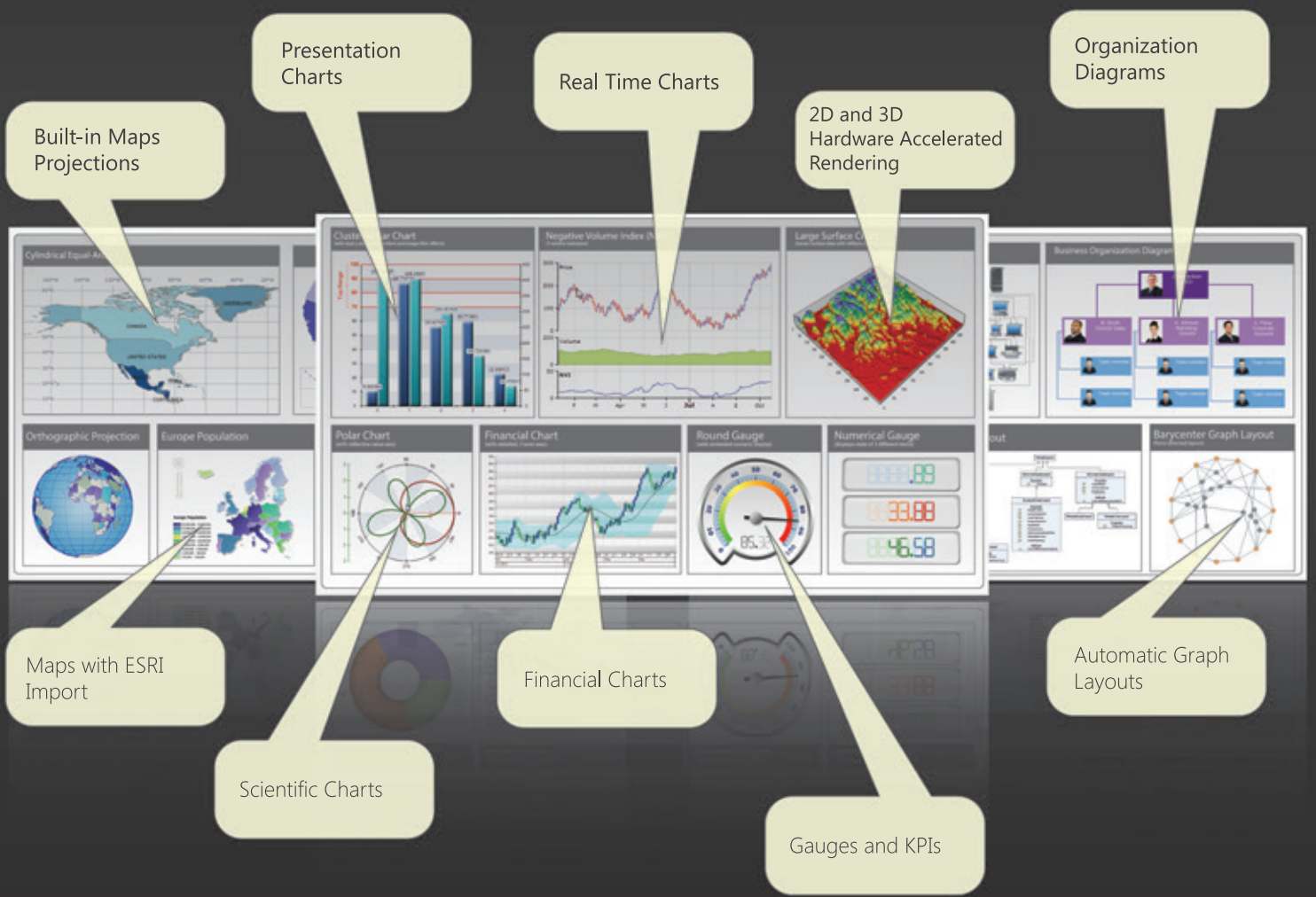
Hadi Brais *is a doctorate scholar at the Indian Institute of Technology Delhi, researching compiler optimizations for the next-generation memory technology. He spends most of his time writing code in C/C++/C# and digging deep into runtimes, compiler frameworks and computer architectures. He blogs at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

Thanks to the following Microsoft technical expert for reviewing this article: *Vance Morrison*

Nevron Data Visualization

The leading data visualization components for desktop and web development in a single package.

NEW 2016 RELEASE COMING SOON!



solutions available for

Microsoft ASP.net | Microsoft Silverlight | WPF | WinForms | SharePoint | Microsoft SQL Server

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

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

Discrete Event Simulation: A Population Growth Example

Arnaldo Pérez Castaño

Throughout history, the ability to simulate has aided the development of multiple sciences. Medical models simulating the human body enhance the study of human anatomy. Computer simulation games such as "World of Warcraft" recreate an entire fantasy world and "Flight Simulator" helps train pilots on the ground. Various simulation programs explore responses to terrorist attacks, pandemic diseases and other possible crises. Even the simulated dinosaurs in the film "Jurassic Park" hint at the broad application of simulation and its potential.

Simulation is a technique in which a real-life system or process is emulated by a designed model. The model encapsulates all of the system's features and behaviors; the simulation is the execution of this system over time. There are several stages to designing a simulation:

- Defining the system to be modeled, which involves studying the problem at hand, identifying the properties of the environment and specifying the goals to reach.
- Formulating the model, which includes defining all of its variables and their logical relations and creating the necessary flow diagrams.
- Defining the data the model will require to produce the desired outcome.
- Producing a computerized implementation of the model.
- Verifying whether the implemented model satisfies the design.
- Validating through comparison that the simulator actually represents the real system being simulated.
- Experimenting to generate desired data using the simulator.
- Analyzing and interpreting results from the simulator and making decisions based on these results.
- Documenting the model created and the simulator as a tool.

Simulations generally comprise either a continuous process or discrete events. To simulate a weather system, for example, the tracking occurs continuously as all elements are constantly changing. Hence, the temperature variable placed against the time variable would be represented by a continuous curve. In contrast, airplane takeoffs or landings occur as points in time and, therefore, a simulation can consider only those precise moments or events and discard everything else. This type of simulation is known as discrete event simulation (DES), and it's what I'll discuss in this article.

This article discusses:

- The main constituents and requirements of a discrete event simulation
- Population elements
- Poisson, exponential and normal distributions
- Developing the simulation application

Technologies discussed:

C#, Object-Oriented Programming

Code download available at:

msdn.com/magazine/0316magcode

Discrete Event Simulation

DES models a system or process as an ordered sequence of individual events over time, that is, from the time of one event to the time of the next event. Hence, in a DES simulation, time is usually much shorter than real time.

When developing a DES, there are six main elements to consider:

Objects represent elements of the real system. They have properties, they relate to events, they consume resources, and they enter and leave queues over time. In the airplane takeoff and landing scenario mentioned earlier, the objects would be airplanes. In a health care system, objects might be patients or organs. In a warehouse system, the objects would be the products in stock. Objects

Figure 1 **Probability of a Relationship Ending**

Average Age	Probability
14-20	0.7
21-28	0.5
29+	0.2

are supposed to interact with each other or with the system and they can be created at any time during a simulation.

Properties are features particular to each object (size, landing time, sex, price and so on) that are stored in order to determine responses to various scenarios that might occur in the simulation; such values can be modified.

Events are things that can occur in the system, especially to objects, such as the landing of an airplane, the arrival of a product at a warehouse, the appearance of a particular disease and so forth. Events can occur and reoccur in any order.

Resources are elements that provide services to objects (for example, a landing strip in an airport, storage cells in a warehouse and doctors in a clinic). When a resource is occupied and an object needs it, the object must queue and wait until the resource is available.

Queues are the conduits in which objects are organized to await the release of a resource that's currently occupied. Queues can have a maximum capacity and they can have different calling approaches: first-in-first-out (FIFO), last-in-first-out (LIFO), or based on some criteria or priority (disease progression, fuel consumption and the like).

Time (as it happens in real life) is essential in simulation. To measure time, a clock is started at the beginning of a simulation and can then be used to track particular periods of time (departure or arrival time, transportation time, time spent with certain symptoms, and so on). Such tracking is fundamental because it allows you to know when the next event should occur.

Because simulation programming can be complicated, there have been many attempts to create languages that embody all the requirements of the simulation paradigm in order to ease development. One such language is SIMULA, invented in the 1960s by Ole Johan and Kristen Nygaard and the first to introduce the concept of object-oriented programming (OOP), today's leading programming paradigm. Nowadays, the focus is more on creating packages, frameworks or libraries that incorporate what programmers need when creating a simulation. These libraries are meant to be called from ordinary languages like C#, C++, Java or Python.

In "A History of Discrete Event Simulation Programming Languages," Nance proposed six minimum requirements any DES programming language should satisfy (bit.ly/1ZnvkVn):

- Random number generation.
- Process transformers, to allow variables other than the uniform random variables.
- List processing, to facilitate the creation, manipulation and deletion of objects.
- Statistical analysis, to provide a descriptive summary of model behavior.
- Report generation, to assist in the presentation of large sets of data and facilitate decision making.
- A time-flow mechanism.

These requirements can all be satisfied in C#, so in this article I'll present a discrete event simulation for population growth developed in C#.

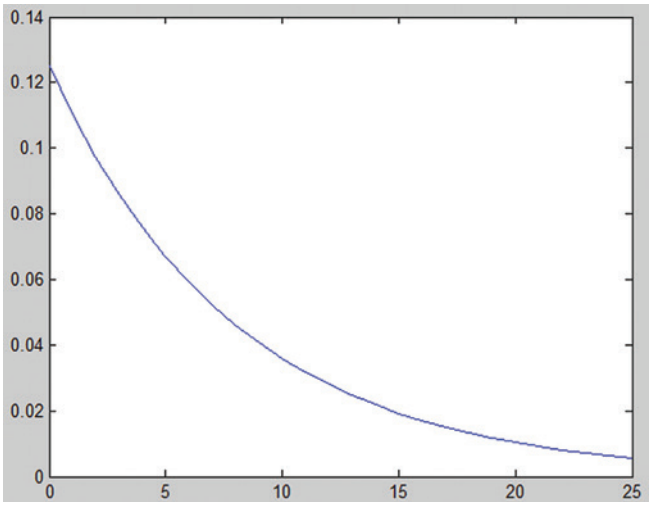


Figure 3 **Exponential Distribution, Parameter λ = 8**

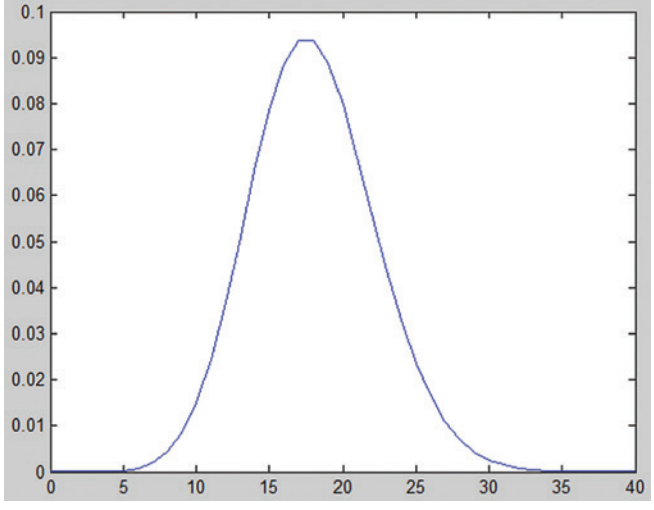


Figure 2 **Poisson Distribution, Parameter λ = 18**

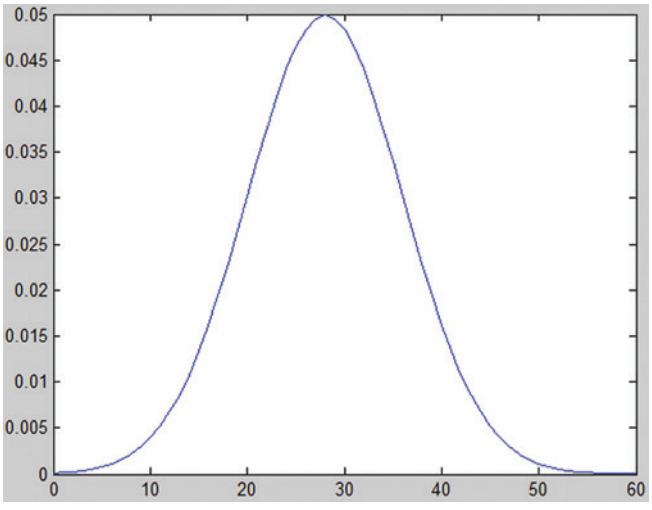


Figure 4 **NormalDistribution, Parameters μ = 28 and σ^2 = 8 Years**

DES for Population Growth

Population growth is one of many aspects considered in the study of how populations (animals and plants) change over time and space and interact with their environment. Populations are groups of organisms of the same species living at the same time, sometimes in the same space or further delimited by particular characteristics.

Why is it important to study population growth? A better understanding of how populations grow or shrink gives scientists the possibility of making better predictions about changes in biodiversity conservation, resource use, climate change, pollution, health care, transportation needs, and so on. It also provides insight into how organisms interact with each other and the environment, a critical aspect when considering whether a population might prosper or decline.

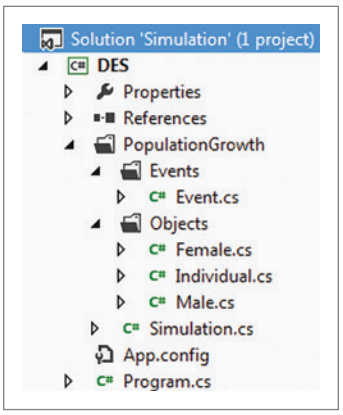


Figure 5 **The Structure of the Simulation Application**

In this article I'll present a DES for the growth of a population. The goal is to observe how the population evolves over time and get some statistics by analyzing the end results (population size, old people, young people and so on). The population will start with *m* male and *n* female individuals, each having an associated age. Clearly, m and n must be greater than zero or the simulation would be pointless. The objects in this model are individuals.

An individual is capable of starting a relationship with another individual after reaching an age that distributes by a Poisson function with parameter λ = 18 years. (You don't need to fully understand Poisson, normal or exponential distributions right now; they'll be explained in the next section.)

There's a less than a 50 percent probability of single and capable opposite-sex individuals engaging with one another, and even that occurs only when the age difference is no greater than 5 years. **Figure 1** shows an example of the probability of a relationship between two individuals ending.

Individuals involved in a relationship can have a child after a time that distributes by an exponential function with parameter λ= 8 years.

A woman can get pregnant if she has an age that follows a normal (bell-shaped) distribution function with parameters μ = 28 and $\sigma^2 = 8$ years. Every woman has a number of children that distributes by a normal function with parameters μ = 2and $\sigma^2 = 6$ years. (The parameter μ represents the average age while σ^2 is the measure of age variability.)

Every individual has a life expectancy that distributes by a Poisson function with parameter λ= 70 years, where λ represents the average life expectancy.

In the previous description it's possible to identify several events:

- Starting a relationship.
- Ending a relationship.
- Getting pregnant.
- Having a child.
- Dying.

Every event is accompanied by a discrete random variable that determines the moment in which the event will occur.

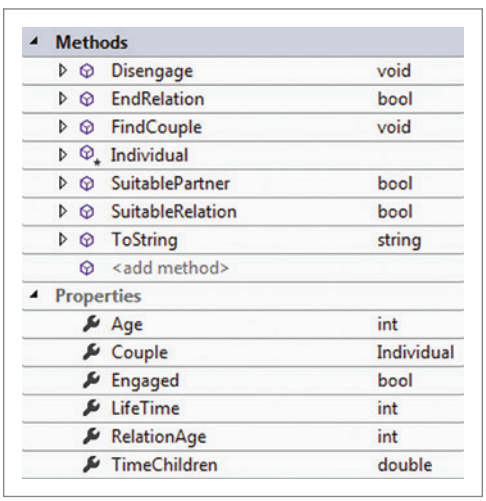


Figure 6 **Properties and Methods of the Individual Class**

Probabilistic Distributions

A discrete random variable is one whose set of values is finite or countably infinite. That is, the values can be listed as a finite or infinite sequence of values 1, 2, 3 …. For a discrete random variable, its probability distribution is any graph, table or formula that assigns a probability to each possible value. The sum of all probabilities must be 1, and each individual probability must be between 0 and 1. For instance, when throwing a fair die (all sides equally probable), the discrete random variable X representing the possible outcomes will have the probabilistic distribution X(1) = 1/6, X(2) = 1/6, …, X(6) = 1/6. All sides are equally probable, so the assigned probability of each is 1/6.

Parameters l and μ indicate the mean (expected value) in their corresponding distributions. The mean represents the value that the random variable takes on average. In other words, it's the sum E=[(each possible outcome) × (probability of that outcome)], where E denotes the mean. In the case of the die, the mean would be E = 1/6 + 2/6 + 3/6 + 4/6 + 5/6 + 6/6 = 3.5. Note that the result 3.5 is actually halfway between all possible values the die can take; it's the expected value when the die is rolled a large number of times.

Parameter σ2 indicates the variance of the distribution. Variance represents the dispersion of possible values of the random variable, and it's always non-negative. Small variances (close to 0) tend to indicate that values are close to each other and the mean; high variances (close to 1) indicate great distance among values and from the mean.

Poisson is a discrete distribution that expresses probabilities regarding the number of events per time unit (see **Figure 2**). It's usually applied when the probability of an event is small and the number of opportunities for it to occur is large. The number of misprints in a book, customers arriving at a business center, cars arriving at traffic lights and deaths per year in a given age group are all examples of applications of the Poisson distribution.

An *exponential* distribution expresses time between events in a Poisson process (see **Figure 3**). For example, if you're dealing with a Poisson process describing the number of customers arriving at a business center during a certain time, you may be interested in a random variable that would indicate how much time passed before the first customer arrived. An exponential distribution can serve this purpose. It could also be applied to

physics processes, for example to represent the lifetime of particles, where λ would indicates the rate at which the particle ages.

The normal distribution describes probabilities that tend to fall around a central value, no bias left or right, as shown in **Figure 4**. Normal distributions are symmetric and possess bell-shaped density curves with a single peak at the mean. Fifty percent of the distribution lies to the left of the mean and 50 percent to the right. The standard deviation indicates the spread or girth of the bell curve; the smaller the standard deviation the more concentrated the data. Both the mean and the standard deviation must be indicated as parameters to the normal distribution. Many natural phenomena closely follow a normal distribution: blood pressure, people's height, errors in measurements and many more.

Now I'll show you how to implement the proposed DES in a popular, sophisticated and elegant language such as C#.

Figure 7 **The FindPartner Method**

```
Public void FindPartner(IEnumerable<Individual> population, int currentTime,
  Dictionary<Event, IDistribution> distributions)
{
foreach (var candidate in population)
if (SuitablePartner(candidate) &&
candidate.SuitableRelation() &&
((ContinuousUniform) distributions[Event.BirthEngageDisengage]).Sample() <= 0.5)
{
// Relate them
candidate.Couple = this;
Couple = candidate;
// Set time for having child
var childTime = ((Exponential)  distributions[Event.TimeChildren]).Sample()*100;
// They can have children on the simulated year: 'currentTime + childTime'.
candidate.TimeChildren = currentTime + childTime;
TimeChildren = currentTime + childTime;
break;
}
}
```

Figure 8 **TheGiveBirth Method**

```
Public Individual GiveBirth(Dictionary<Event, IDistribution>
distributions, int currentTime)
  {
var sample =
  ((ContinuousUniform) distributions[Event.BirthEngageDisengage]).Sample();
var child = sample > 0.5 ? (Individual) newMale(0) : newFemale(0);

// One less child to give birth to
ChildrenCount--;

child.LifeTime = ((Poisson)distributions[Event.Die]).Sample();
child.RelationAge = ((Poisson)distributions[Event.CapableEngaging]).Sample();

if (child is Female)
{
(child as Female).PregnancyAge =
  ((Normal)distributions[Event.GetPregnant]).Sample();
(child as Female).ChildrenCount =
  ((Normal)distributions[Event.ChildrenCount]).Sample();
}

if (Engaged && ChildrenCount> 0)
{
TimeChildren =
  currentTime + ((Exponential)distributions[Event.TimeChildren]).Sample();
Couple.TimeChildren = TimeChildren;
}
else
TimeChildren = 0;

IsPregnant = false;
return child;
  }
```

Implementation

To develop this simulation I'll exploit all the benefits of the OOP paradigm. The idea is to obtain the most readable code possible. Scientific applications tend to be complex and difficult to understand, and it's essential to try to make them as clear as possible so that others can comprehend your code. I'll develop the application as a console application in C#, with the structure shown in **Figure 5**.

A logical path is essential; note how the namespace structure maintains its own common sense: Simulation.DES.PopulationGrowth.

The Events folder contains an Event.cs file, which defines an enum type representing all possible events in the simulation:

```
Namespace DES.PopulationGrowth.Events
{
Public enum Event
  {
Capable Engaging,
Birth EngageDisengage,
GetPregnant,
ChildrenCount,
TimeChildren,
    Die
  }
}
```

The Objects folder contains every class related to individuals; these are the pieces of code that will benefit most from OOP. There are three classes related to individuals: Individual, Male and Female. The first is an abstract class and the others inherit from it; that is, males and females are individuals. Most of the coding happens in the Individual class. **Figure 6** shows its properties and methods.

Here's a description of every property:

- Age: Individual age.
- Couple: Null if individual is single, otherwise his/her couple, another Individual.
- Engaged: True if individual is involved in a relation, false otherwise.
- LifeTime: Age at which the individual dies.
- RelationAge: Age at which the individual can start a relationship.
- TimeChildren: Time in the simulation (not age) at which the individual can have children.

And here's a description of their methods:

- Disengage: Ends the relation between two individuals.
- EndRelation: Determines whether the relation should end, according to the probability table in **Figure 1**.
- FindPartner: Finds an available partner for an individual.
- SuitablePartner: Determines whether an individual is suitable for starting a relationship (age difference and opposite sex).
- SuitableRelation: Determines whether some individual can start a relationship; that is, is single and at an age for starting a relation.
- ToString: Override to represent individual information as a string.

The class starts by defining all properties and later the constructor, which merely sets the age:

```
Public abstract class Individual
  {
Public int Age { get; set; }
Public int RelationAge{ get; set; }
Public int LifeTime{ get; set; }
Public double TimeChildren{ get; set; }
Public Individual Couple { get; set; }

protected Individual(int age)
{
  Age = age;
}
```

Figure 9 **The Simulation Class Constructor**

```
public Simulation(IEnumerable<Individual> population, int time)
{
Population = new List<Individual>(population);
Time = time;
_distributions = new Dictionary<Event, IDistribution>
{
{ Event.CapableEngaging, new Poisson(18) },
{ Event.BirthEngageDisengage, new ContinuousUniform() },
{ Event.GetPregnant, new Normal(28, 8) },
{ Event.ChildrenCount, new Normal(2, 6) },
{ Event.TimeChildren, new Exponential(8) },
{ Event.Die, new Poisson(70) },
                                };

foreach (var individual in Population)
{
// LifeTime
individual.LifeTime = ((Poisson) _distributions[Event.Die]).Sample();
// Ready to start having relations
individual.RelationAge = ((Poisson)_distributions[Event.CapableEngaging]).Sample();
// Pregnancy Age (only women)
if (individual is Female)
                  {
(individual as Female).PregnancyAge = ((Normal) _distributions[Event.
GetPregnant]).Sample();
(individual as Female).ChildrenCount = ((Normal)_distributions[Event.
ChildrenCount]).Sample();
}
}
}
```

Figure 10 **The Execute Method**

```
Public void Execute()
{
while (_currentTime< Time)
{
// Check what happens to every individual this year
for (vari = 0; i<Population.Count; i++)
{
var individual = Population[i];

// Event -> Birth
if (individual is Female&& (individual as Female).IsPregnant)
Population.Add((individual as Female).GiveBirth(_distributions,
  _currentTime));

// Event -> Check whether someone starts a relationship this year
if (individual.SuitableRelation())
individual.FindPartner(Population, _currentTime, _distributions);

// Events where having an engaged individual represents a prerequisite
if (individual.Engaged)
{
// Event -> Check whether arelationship ends this year
if (individual.EndRelation(_distributions))
individual.Disengage();

// Event -> Check whether a couple can have a child now
if (individual is Female &&
(individual as Female).SuitablePregnancy(_currentTime))
(individual as Female).IsPregnant = true;
}

// Event -> Check whether someone dies this year
if (individual.Age.Equals(individual.LifeTime))
{
// Case: Individual in relationship (break relation)
if (individual.Engaged)
individual.Disengage();
Population.RemoveAt(i);
}

individual.Age++;
_currentTime++;

}
}
```

The SuitableRelation and SuitablePartner methods and the Engaged property are just logical tests, pretty straightforward:

```
Public bool SuitableRelation()
{
return Age >= RelationAge&& Couple == null;
}

Public bool SuitablePartner(Individual individual)
{
return ((individual is Male && this is Female) ||
(individual is Female && this is Male)) &&Math.Abs(individual.Age - Age) <= 5;
}

Public bool Engaged
{
get { return Couple != null; }
}
```

The Disengage method breaks up a relationship by setting the Couple field to null on the couple and later on the individual. It also sets their time for having children to 0 because they're no longer engaged.

```
Public void Disengage()
{
Couple.Couple = null;
Couple = null;
TimeChildren = 0;
}
```

The EndRelation method is basically a translation of the probability table for determining the chance of a relationship ending. It uses a uniform random variable, which produces a random value in the range [0, 1] equivalent to producing an acceptance percentage. The distributions dictionary is created in the simulation class (described shortly) and holds pairs (event, probability distribution), thus associating every event with its distribution:

```
Public bool EndRelation(Dictionary<Event, IDistribution> distributions)
{
var sample =
  ((ContinuousUniform) distributions[Event.BirthEngageDisengage]).Sample();

if (Age >= 14 && Age <= 20 && sample <= 0.7)
return true;
if (Age >= 21 && Age <= 28 && sample <= 0.5)
return true;
if (Age >= 29 && sample <= 0.2)
return true;

return false;
}
```

The FindPartner method, shown in **Figure 7**, finds an available partner for the instance individual. It receives as input the population list, the current time of the simulation and the distributions dictionary.

Last, the ToString method defines the string representation of an individual:

```
Public override string ToString()
{
Return string.Format("Age: {0} Lifetime {1}", Age, LifeTime);
}
The Male class is very simple:
Public class Male: Individual
{
public Male(int age) : base(age)
{
}

Public override string ToString()
{
Return base.ToString() + " Male";
}
}
```

The Female class is a bit more complicated because it includes treatment for pregnancy, birth and so forth. The class definition starts by declaring all properties and the constructor:

Figure 11 **Viewing the Population over Time**

```
Static void Main()
{
var population = new List<Individual>
{
New Male(2),
New Female(2),
New Male(3),
New Female(4),
New Male(5),
New Female(3)
};

var sim = new Simulation(population, 1000);
sim.Execute();

// Print population after simulation
foreach (var individual in sim.Population)
Console.WriteLine("Individual {0}", individual);

Console.ReadLine();
}
```

```
Public class Female :Individual
{
Public bool IsPregnant{ get; set; }
Public double PregnancyAge{ get; set; }
Public double ChildrenCount{ get; set; }

public Female(int age) : base(age)
{
}
}
```

Here are the properties of the Female class:

- IsPregnant: Determines whether this woman is pregnant.
- PregnancyAge: Determines age at which a woman can get pregnant.
- ChildrenCount: Indicates number of children to give birth to.

And here are the methods it contains:

- SuitablePregnancy: Determines whether this woman can get pregnant.
- GiveBirth: Indicates a woman giving birth, adding new individual to the population.
- ToString:override: Used to represent female information as a string.

SuitablePregnancy is a test method that outputs true when the instance woman satisfies every condition for getting pregnant:

```
Public bool SuitablePregnancy(intcurrentTime)
  {
return Age >= PregnancyAge && currentTime <= TimeChildren && ChildrenCount > 0;
}
```

The GiveBirth method, shown in **Figure 8**, is the code that adds and initializes new individuals into the population.

A uniform sample is generated to first determine if the new individual will be female or male, each with a probability of 50 percent (0.5). The ChildrenCount value is decremented by 1, indicating that this woman has one less child to give birth to. The rest of the code relates to individual initialization and resetting some variables.

The ToString method changes the manner in which Females are represented as strings:

```
Public override string ToString()
  {
Return base.ToString() + " Female";
}
```

Because all functions relating to individuals were placed in separate classes, the Simulation class is now much simpler. Properties and the constructor are at the beginning of the code:

```
Public class Simulation
  {
Public List<Individual> Population { get; set; }
Public int Time { get; set; }
Private int _currentTime;
Private readonly Dictionary<Event, IDistribution> _distributions ;
```

The properties and variables are self-descriptive and some were previously described. Time represents the time (in years) the simulation will last, and _currentTime represents the current year of the simulation. The constructor in this case, shown in **Figure 9**, is more complicated because it includes the initialization of random variables for each individual.

Finally, the Execute method, shown in **Figure 10**, is where all simulation logic occurs.

Iterations in the outer loop represent years that go by in the simulation. The inner loop goes through events that might occur to those individuals in that particular year.

To see how the population evolves over time, I set up a new console application, shown in **Figure 11**.

Figure 12 shows the population after 1,000 years.

In this article I developed a discrete event simulation to see how a population evolves in time. The object-oriented approach proved very useful for obtaining readable, concise code that readers can try and improve if necessary. ■

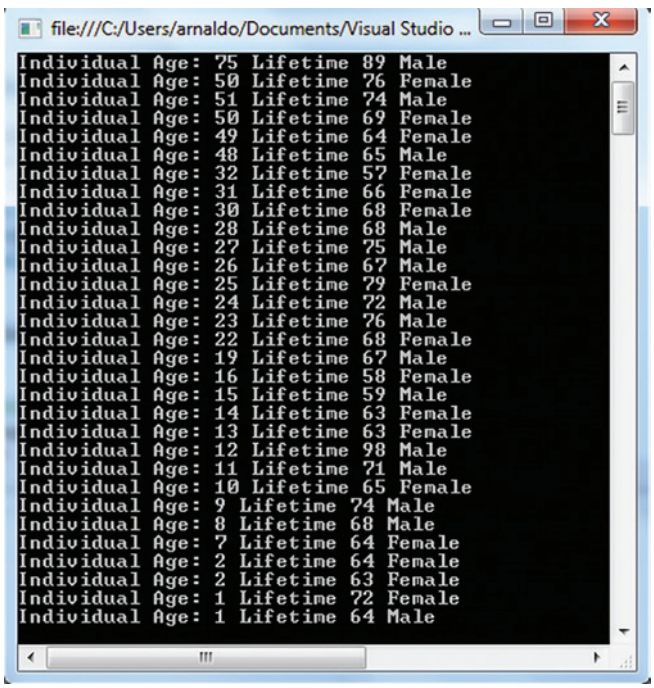


Figure 12 **Population After 1,000 Years**

Arnaldo Pérez Castaño *is a computer scientist based in Cuba. He is the author of a series of programming books—"JavaScript Fácil," "HTML y CSS Fácil," and "Python Fácil" (Marcombo S.A). His expertise includes Visual Basic, C#, .NET Framework, and Artificial Intelligence, and offers his services as a freelancer through nubelo.com. Cinema and music are some of his passions. Contact him at arnaldo.skywalker@gmail.com.*

Thanks to the following Microsoft technical expert who reviewed this article: *James McCaffrey*

CAMPAIGN FOR CODE 2016 ★ VISUAL STUDIO LIVE!

Las Vegas CODE

WE CAN BELIEVE IN

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

LAS VEGAS 2016 CAMPAIGN FOR CODE

VISUAL STUDIO LIVE! is on a Campaign for Code in 2016, in support of developer education. First up on the campaign trail? Fabulous Las Vegas, where developers, software architects, engineers, designers and more will convene for five days of unbiased and cutting-edge education on the Microsoft Platform. Sharpen your skills in Visual Studio, ASP.NET, JavaScript, HTML5, Mobile, Database Analytics, and so much more.

EVENT PARTNERS Microsoft Magenic

PLATINUM SPONSOR Alachisoft

SUPPORTED BY Visual Studio msdn magazine Visual Studio MAGAZINE

PRODUCED BY 1105 MEDIA YOUR GROWTH. OUR BUSINESS.

LAS VEGAS • MARCH 7-11, 2016

BALLY'S HOTEL & CASINO

TUESDAY KEYNOTE:

The Future of Application Development—Visual Studio 2015 and .NET 2015

Jay Schmelzer
Director of Program Management,
Visual Studio Team, Microsoft

WEDNESDAY GENERAL SESSION:

Featuring ASP.NET Core 1.0

Scott Hunter
Principal Program Manager Lead,
ASP.NET Team, Microsoft

DEVELOPMENT TOPICS INCLUDE:

- ALM / DevOps
- ASP.NET
- Cloud Computing
- Database & Analytics
- JavaScript / HTML5 Client
- Mobile Client
- Modern App Development
- UX / Design
- Visual Studio / .NET
- Windows Client

Be a responsible dev citizen. Register to code with us today!

SESSIONS ARE FILLING UP QUICKLY!

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

USE PROMO CODE VSLMAR2

CONNECT WITH VISUAL STUDIO LIVE!

twitter.com/vslive – @VSLive | facebook.com – Search "VSLive" | linkedin.com – Join the "Visual Studio Live" group!

VSLIVE.COM/LASVEGAS

Introduction to SciPy Programming for C# Developers

James McCaffrey

There's no formal definition of the term data science, but I think of it as using software programs to analyze data using classical statistics techniques and machine learning algorithms. Until recently, much of data science analysis was performed with expensive commercial products, but in the past few years the use of open source alternatives has increased greatly.

Based on conversations with my colleagues, the three most common open source approaches for data science analysis are the R language, the Python language combined with the SciPy ("scientific Python") library, and integrated language and execution environments such as SciLab and Octave.

In this article I'll give you a quick tour of programming with SciPy so you can understand exactly what it is and determine if you want to spend time learning it. This article assumes you have some experience with C# or a similar general-purpose programming language such as Java, but doesn't assume you know anything about Python or SciPy.

In my opinion, the most difficult part about learning a new programming language or technology is just getting started, so I'll describe in detail how to install (and uninstall) the software needed to run a SciPy program. Then, I'll describe several ways to edit and execute a SciPy program and explain why I prefer using the Integrated Development Environment (IDLE) program.

I'll conclude by walking you through a representative program that uses SciPy to solve a system of linear equations, in order to demonstrate similarities and differences with C# programming. **Figure 1** shows the output of the demo program and gives you an idea of where this article is headed.

This article discusses:

- Installing the SciPy stack
- Installing NumPy and SciPy
- Editing and running a SciPy program
- A SciPy demo program

Technologies discussed:

C#, Python, SciPy, IDLE

Code download available at:

msdn.com/magazine/0316magcode

Installing the SciPy Stack

The SciPy stack has three components: Python, NumPy and SciPy. The Python language has basic features such as while loop control structures and a general-purpose list data type, but interestingly, no built-in array type. The NumPy library adds support for arrays and matrices, plus some relatively simple functions such as array search and array sort. The SciPy library adds intermediate and advanced functions that work with data stored in arrays and matrices.

To run a SciPy program (technically a script because Python is interpreted rather than compiled), you install Python, then

NumPy, then SciPy. Installation isn't too difficult, and you can install a software bundle that includes all three components. One common bundle is the Anaconda distribution, which is maintained by Continuum Analytics at continuum.io. However, I'll show you how to install the components individually.

Python is supported on nearly all versions of Windows. To install Python, go to python.org/downloads, where you'll find the option to install either a Python 3.x version or a 2.x version (see **Figure 2**). The two versions aren't fully compatible, but the NumPy and SciPy libraries are supported on both. I suggest installing the 2.x version because there are some third-party functions that aren't yet supported on the 3.x version.

When you click a download button, you'll get the option to either run the .msi installer program immediately or save it so you can run it later. You can click the Run button. The installer uses a wizard. The first screen asks if you want to install for all users or just the current user. The default is for all users so click the Next button.

The next screen asks you to specify the root installation directory. The default is C:\Python27 (rather than the more usual C:\Program Files directory) and I suggest you use the default location and click Next. The following screen lets you include or exclude various features such as documentation and utility tools like pip ("pip installs Python"). The default Python features are fine, so click on the Next button.

The installation starts and you'll see a window with a familiar blue progress bar. When installation finishes, you'll see a window with a Finish button. Click on that button.

By default, the Python installation process doesn't modify your machine's PATH environment variable. You'll want to add C:\Python27, C:\Python27\Scripts and C:\Python27\Lib\idlelib to the PATH variable so you can run Python from a command shell and launch the IDLE editor without having to navigate to their directory locations.

You should verify that Python is installed correctly. Launch a command shell and navigate to your system root directory by entering a cd \ command. Now enter the command python -- version (note the two dash characters). If Python responds, it's been successfully installed.

Installing NumPy and SciPy

It's possible to install the NumPy and SciPy packages from source code using the Python pip utility. The pip approach works well with packages that are pure Python code, but NumPy and SciPy have hooks to compiled C language code, so installing them using pip is quite tricky.

Luckily, members of the Python community have created pre-compiled binary installers for NumPy and SciPy. The ones I recommend using are maintained in the SourceForge repository. To install NumPy, go to bit.ly/1Q3mo4M, where you'll see links to various versions. I recommend using the most recent version that has the most download activity.

You'll see a list of links. Look for a link with a name that resembles numpy-1.10.2-win32-superpack-python2.7.exe, as shown in **Figure 3**. Make sure you have the executable that corresponds to your version of Python and click on that link. After a brief delay you'll get an option to run the self-extracting executable installer immediately, or save it to install later. Click on the Run button.

The NumPy installer uses a wizard. The first screen just shows an introductory splash window. Click the Next button. The next screen asks you to specify the installation directory. The installer will find

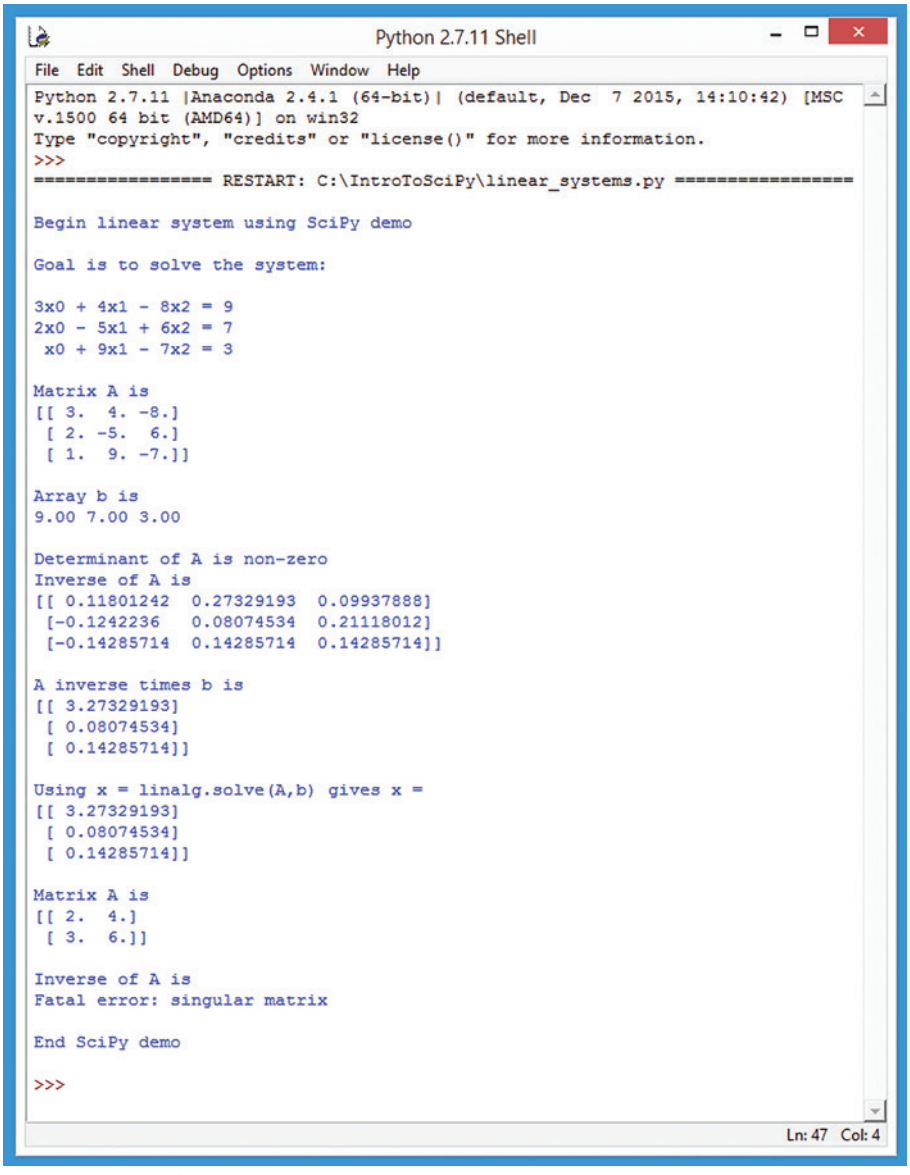


Figure 1 **Output from a Representative SciPy Program**

your existing Python installation and recommend installing NumPy in the C:\Python27\Lib\site-packages directory. Accept this location and click Next.

The next screen gives you a last chance to back out of the install, but don't do so. Click the Next button. You'll see a progress window during installation and if you watch closely, you'll see some interesting logging messages. When the NumPy installation finishes, you'll be presented with a Finish button. Click on it. Then you'll see a final Setup Completed window with a Close button. Click on it.

> One nice characteristic of the SciPy stack is that it's very easy to uninstall components.

After you've installed NumPy, the next step is to install the SciPy package, which is identical to installing NumPy. Go to bit.ly/1QbwJOz and find a recent, well-used directory. Go into that directory and find a link to an executable with a name like scipy-0.16.1-win32-superpack-python2.7.exe and click on it to launch the self-extracting executable installer.

One nice characteristic of the SciPy stack is that it's very easy to uninstall components. You can go to the Windows Control Panel, Programs and Features, select the component (that is, Python, or NumPy, or SciPy) to remove and then click the Uninstall button, and the component will be quickly and cleanly removed.

Editing and Running a SciPy Program

If you write programs using a .NET language, there aren't many options available and you almost certainly use Visual Studio. But when writing a Python program you have many options. I recommend using the IDLE editor and execution environment.

The idle.bat program launcher file is located by default in the C:\Python27\Lib\idelib directory. If you added this directory to your system PATH environment variable, you can launch IDLE by opening a command shell and entering the command idle. This will start the IDLE Python Shell program as shown in the top part of **Figure 4**.

You can create a new Python source code file by clicking on the File | New File item on the menu bar. This opens a similar-looking separate editor window as shown in the bottom part of **Figure 4**. Type these seven statements in the editor window:

```
# test.py
import numpy as np
import scipy.misc as sm
arr = np.array([1.0, 3.0, 5.0])
print arr
n = sm.factorial(4)
print "4! = " + str(n)
```

Then save your program as test.py in any convenient directory. Now you can run your program by clicking on the Run | Run Module menu item in the editor window, or by hitting the F5 shortcut key. Program output will be displayed in the Python Shell window. Simple!

Some experienced Python developers take potshots at IDLE because it is rather simple. But that's exactly why I like it. You don't get anything near the sophisticated programming environment of Visual Studio, but you do get syntax coloring and a good error message generator when you've written incorrect code.

Instead of using IDLE to edit and run programs, you can use any text editor, including Notepad, to write and save a Python program. Then you can execute the program from a command line like this:

```
C:\IntroToPython> python test.py
```

This assumes you have the path to the python.exe interpreter in your system PATH environment variable. Output will be displayed in the command shell.

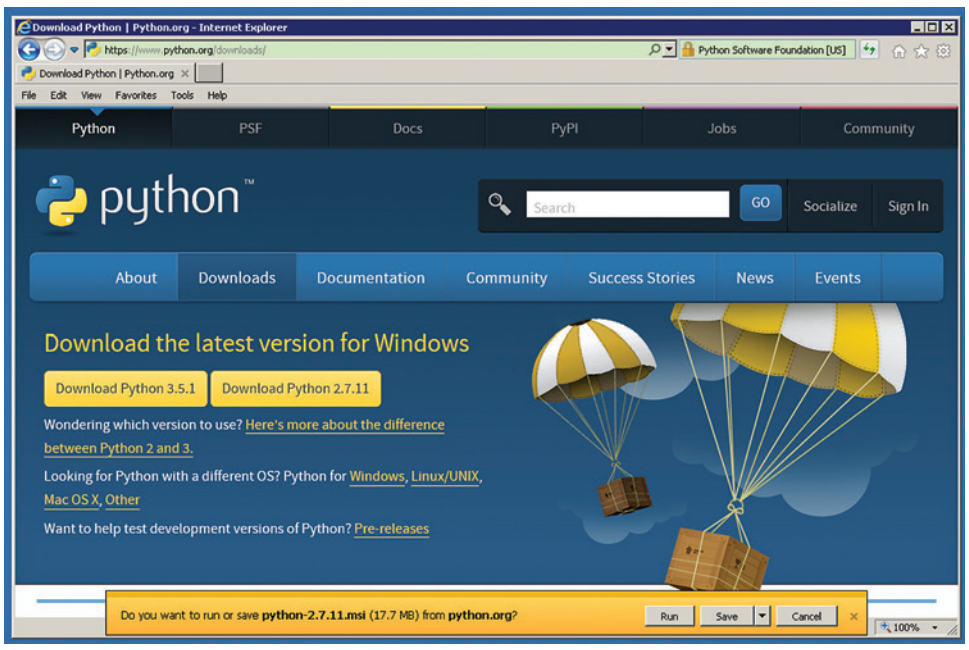


Figure 2 **Installing Python**

There are many Python IDEs. One popular open source IDE that's specifically intended for use with SciPy is the Scientific Python Development Environment (Spyder) program. You can find information about it at pythonhosted.org/spyder.

An interesting alternative to IDLE and Spyder is the open source Python Tools for Visual Studio (PTVS) plug-in. As the name implies, PTVS allows you to edit and run Python programs using Visual Studio. You can find information about PTVS at microsoft.github.io/PTVS.

A SciPy Demo Program

Take a look at the Python program in **Figure 5**, or better yet, type or download the file that accompanies this article into a Python editor and run the program. The demo is

FREE TRIAL

DocuVieware

Universal HTML5 Viewer & Document Management Kit

supports nearly 100 file formats

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

HOT FEATURES IN THE NEW MAJOR VERSION

- Office Open XML (.docx) support
- TWAIN acquisition
- PDF form fields interactions
- Custom snap-ins
- Higher speed and lower memory usage

Download the **Free Trial** and check the **Online Demos** at www.docuvieware.com

not intended to be a comprehensive set of SciPy examples, but it is designed to give you a good feel for what SciPy programming is like.

The demo program begins with two comment lines:

```
# linear_systems.py
# Python 2.7
```

Because Python 2.x and 3.x versions are not fully compatible, it's not a bad idea to be explicit about which version of Python you used. Next, the demo loads the entire NumPy module and one SciPy sub-module:

```
import numpy as np
import scipy.linalg as spla
```

You can think of these statements as somewhat like adding a reference from a Microsoft .NET Framework DLL to a C# program and then bringing the assembly into scope with a using statement. The linalg sub-module stands for linear algebra. SciPy is organized into 16 primary sub-modules plus two utility sub-modules. Next, the demo implements a program-defined function to display an array:

```
def my_print(arr, cols, dec, nl):
  n = len(arr)
  fmt = "%." + str(dec) + "f" # like %.4f
  for i in xrange(n):  # alt: for x in arr
    if i > 0 and i % cols == 0:
      print ""
    print fmt % arr[i],
  if nl == True:
    print "\n"
```

Python uses indentation rather than curly brace characters to delimit code blocks. Here, I use two spaces for indentation to save space; most Python programmers use four spaces for indentation.

Function my_print has four parameters: an array to display, the number of columns to display the values, the number of decimals for each value and a flag indicating whether to print a newline. The len function returns the size (number of cells) of the array. An alternative is to use the array size property:

```
n = arr.size
```

The xrange function returns an iterator and is the standard way to traverse an array. An alternative is to use a "for x in arr" pattern, which is similar to the C# foreach statement.

> When writing a Python program you have many options. I recommend using the IDLE editor and execution environment.

Because both Python and C# have roots in the C language, much of Python syntax is familiar to C# programmers. In the demo, % is the modulo operator, but it's also used for formatting floating point value output; and is used as a logical operator rather than &&; == is a check for equality; and True and False (capitalized) are Boolean constants.

Next, the demo creates a program-defined function named main, which starts with some print statements that explain the problem to solve:

```
def main():
  print "\nBegin linear system using SciPy demo \n"
  print "Goal is to solve the system: \n"
  print "3x0 + 4x1 - 8x2 = 9"
  print "2x0 - 5x1 + 6x2 = 7"
  print " x0 + 9x1 - 7x2 = 3"
```

The goal is to find values for variables x0, x1 and x2 so that all three equations are satisfied. The name main is not a Python keyword, so it could have been called anything. Having a main function of some sort isn't required. For short programs (typically less than one page of code), I usually dispense with a main function and just start with executable statements.

Next, the demo program sets up the problem by putting the coefficient values into a NumPy 3x3 matrix named A and the constants into a NumPy array named b:

```
A = np.matrix([[3.0, 4.0, -8.0],
               [2.0, -5.0, 6.0],
               [1.0, 9.0, -7.0]])
b = np.array([9.0, 7.0, 3.0])
```

The matrix and array functions here actually accept Python lists (indicated by square brackets) with hardcoded values as their arguments. You can also create matrices and arrays using the NumPy zeros function, and you can read data from a text file into a matrix or an array using the loadtxt function.

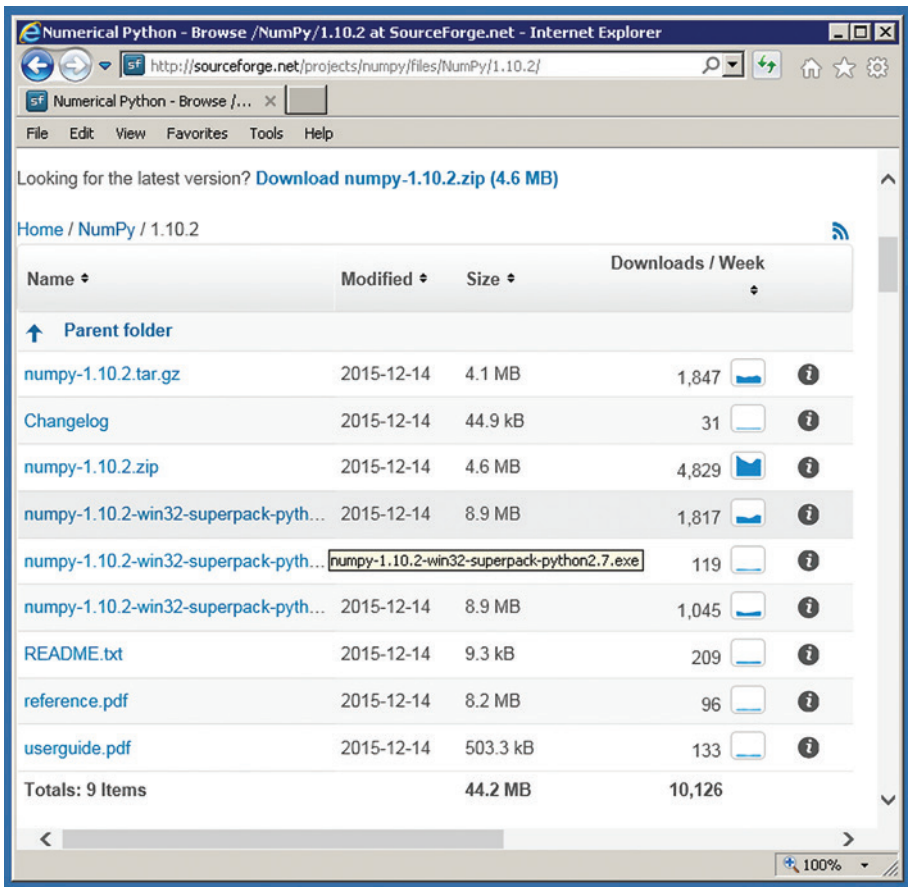


Figure 3 **Installing NumPy**

Work With Relational Data, Not Complex APIs or Services

Whether you are a developer building Apps with ADO.NET, JDBC, OData, or Xamarin, or a systems integrator connecting systems with SQL SSIS or BizTalk, or even an information worker using ODBC or Excel – our state-of-the-art drivers give you live and direct access to any data source. Get simple, standard SQL access to Salesforce, Microsoft Dynamics, MongoDB, BigQuery, Cassandra, SAP, NetSuite, Google Analytics, SharePoint, and much more.

Download a free trial today: www.cdata.com

cdata
SEE THE WORLD AS A DATABASE

If you took an algebra class, you might remember that to solve a system of equations Ax = b for x, where A is a square matrix of coefficients and b is a column matrix (that is, n rows but only 1 column) of the constants, you must find the matrix inverse of A and then matrix-multiply the inverse times column matrix b.

At this point in the demo, b is an array with three cells rather than a 3x1 column matrix. To convert b into a column matrix, the demo program uses the reshape function:

```
b = np.reshape(b, (3,1))
```

The NumPy library has many functions that can manipulate arrays and matrices. For example, the flatten function will convert a matrix to an array. Now, as it turns out, the SciPy matrix multiplication function is smart enough to infer what you intend if you multiply a matrix and an array so the call to reshape isn't really necessary here.

Next, the demo program displays the values in matrices A and b:

```
print "Matrix A is "
print A
print ""
print "Array b is "
my_print(b, b.size, 2, True)
```

In Python 2.x, print is a statement rather than a function, as it is in Python 3.x, so parentheses are optional. The program-defined my_print function doesn't return a value, so it's equivalent to a void C# function and is called as you might expect. Python supports named-parameter calls so the call could've been:

```
my_print(arr=b, cols=3, dec=2, nl=True)
```

Next, the demo program finds the inverse of matrix A:

```
d = spla.det(A)
if d == 0.0:
  print "Determinant of A is zero so no solution "
else:
  Ai = spla.inv(A)
  print "Determinant of A is non-zero "
  print "Inverse of A is "
  print Ai
```

The SciPy det function returns the determinant of a square matrix. If a matrix of coefficients for a system of linear equation has a determinant equal to zero, the matrix can't be inverted. The Python if-else statement should look familiar to you. Python has a neat "elif" keyword for if-else-if control structures, for example:

```
if n < 0:
  print "n is negative"
elif n == 0:
  print "n equals zero"
else:
  print "n is positive"
```

> Some experienced Python developers take potshots at IDLE because it is rather simple. But that's exactly why I like it.

Next, the demo solves the system of equations using matrix multiplication via the NumPy dot function:

```
Aib = np.dot(Ai, b)
print "A inverse times b is "
print Aib
```

The dot function is so named because matrix multiplication is a form of what's called the dot product.

Next, the demo program solves the system of equations directly, using the NumPy solve function:

```
x = spla.solve(A, b)
print "Using x = linalg.solve(A,b) gives x = "
print x
```

Many SciPy and NumPy functions have optional parameters with default values, which is somewhat equivalent to C# method overloading. The SciPy solve function has five optional parameters. The point is that when you see a SciPy or NumPy example function call, even if you think you understand the example, it's a good idea to take a look at the documentation to see if there are any useful optional parameters.

There's some overlap between the NumPy and SciPy libraries. For example, the NumPy package also has linalg sub-module that has a solve function. However, the NumPy solve function has no optional parameters.

Next, the demo program shows an example of the Python try-except mechanism:

```
try:
  A = np.array([[2.0, 4.0], [3.0, 6.0]])
  Ai = spla.inv(A)
  print Ai
except Exception, e:
  print "Fatal error: " + str(e)
```

This pattern should look familiar to you if you've ever used the C# try-catch

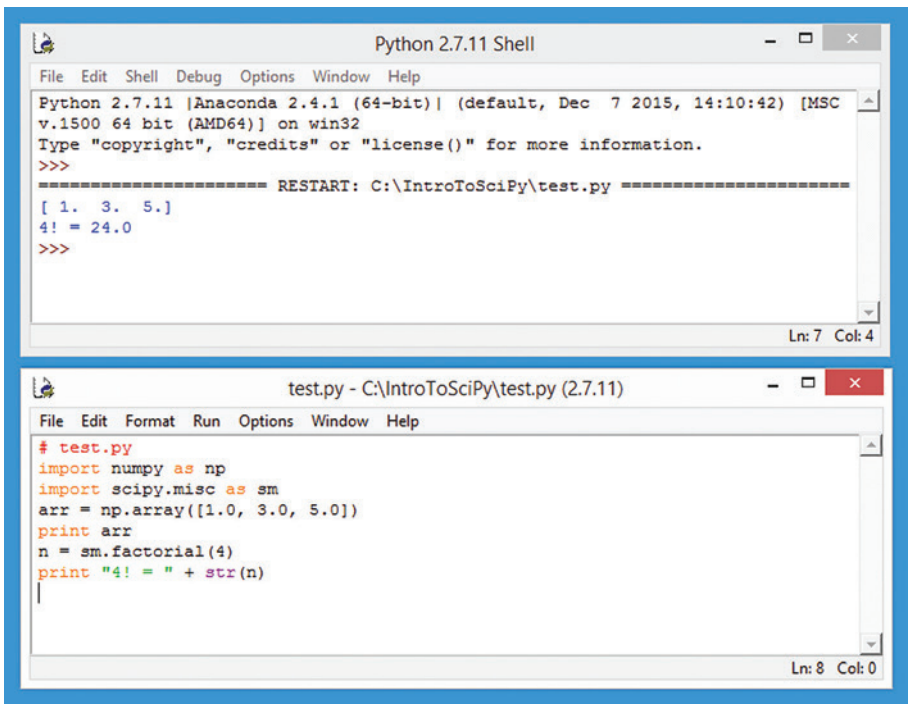


Figure 4 **Editing and Running a Program Using IDLE**

RAVENDB

Safe by Default, Optimized for Efficiency

RavenDB Just Works!

Maximize automated operations, increase efficiency, for an overall performance gain

RavenDB features one of the most **robust replication** mechanisms in the industry, with automatic cluster topology configuration. It also possesses built-in heuristics that transparently configure load patterns in real production time, recognizing high load demand conditions. It will automatically shift resources between background processes and client requests, balancing loads and latency in real time to maximize overall performance, without requiring human intervention.

RavenDB is the all in one self-optimized solution.

www.ravendb.net

Proudly Developed by

Hibernating Rhinos

Figure 5 **A Representative SciPy Program**

```
# linear_systems.py
# Python 2.7

import numpy as np
import scipy.linalg as spla

def my_print(arr, cols, dec, nl):
  n = len(arr)
  fmt = "%." + str(dec) + "f" # like %.4f
  for i in xrange(n):  # alt: for x in arr
    if i > 0 and i % cols == 0:
      print ""
    print fmt % arr[i],
  if nl == True:
    print "\n"

def main():
  print "\nBegin linear system using SciPy demo \n"
  print "Goal is to solve the system: \n"
  print "3x0 + 4x1 - 8x2 = 9"
  print "2x0 - 5x1 + 6x2 = 7"
  print " x0 + 9x1 - 7x2 = 3"
  print ""

  A = np.matrix([[3.0, 4.0, -8.0],
                 [2.0, -5.0, 6.0],
                 [1.0, 9.0, -7.0]])
  b = np.array([9.0, 7.0, 3.0])      # b is an array
  b = np.reshape(b, (3,1))           # b is a col vector

  print "Matrix A is "
  print A
  print ""

  print "Array b is "
  my_print(b, b.size, 2, True)

  d = spla.det(A)
  if d == 0.0:
    print "Determinant of A is zero so no solution "
  else:
    Ai = spla.inv(A)
    print "Determinant of A is non-zero "
    print "Inverse of A is "
    print Ai
    print ""

  Aib = np.dot(Ai, b)
  print "A inverse times b is "
  print Aib
  print ""

  x = spla.solve(A, b)
  print "Using x = linalg.solve(A,b) gives x = "
  print x
  print ""

  try:
    A = np.array([[2.0, 4.0], [3.0, 6.0]])
    print "Matrix A is "
    print A
    print ""
    print "Inverse of A is "
    Ai = spla.inv(A)
    print Ai
  except Exception, e:
    print "Fatal error: " + str(e)

  print "\nEnd SciPy demo \n"

if __name__ == "__main__":
  main()
```

statements. In C#, when you concatenate strings, you can do so implicitly. For example, in C# you could write:

```
int n = 99;
Console.WriteLine("The value of n is " + n);
```

But when you concatenate strings in Python, you must do so explicitly with a cast using the str function:

```
n = 99
print "The value of n is " + str(n)
```

The demo program concludes with a print statement and a special Python incantation:

```
print "\nEnd SciPy demo \n"
if __name__ == "__main__":
  main()
```

> Python uses indentation rather than curly brace characters to delimit code blocks.

The last statement of the demo program could have been just main() ,which would be interpreted as an instruction to call program-defined function main, and the program would run fine. Adding the if __name__ == "__main__" pattern (note that there are two underscore characters before and after name and main) establishes the current module as the program entry point. When a Python program begins execution, the interpreter internally labels the initial module as:

```
"__main__"
```

So, suppose you had some other program-defined modules with executable statements and you imported them. Without the if-check, the Python interpreter would see executable statements in the imported modules and execute them. Put slightly differently, if you add the if-check to your program-defined Python files, these files can be imported by other Python programs and won't cause trouble.

So, What's the Point?

Your initial reaction to this article could well be something like, "Well, this is all somewhat interesting, but in my day-to-day job I really don't need to solve systems of linear equations or use obscure math functions." My response would be, "Well, that's true, but perhaps one of the reasons you don't use some of the functionality of the SciPy library is that you're not aware of what types of problems you can solve."

Put another way, in my opinion, developers tend to tackle problems for which they have the tools. For example, if you know Windows Communication Foundation (WCF) then you'll use WCF (and have my sympathy). If you add SciPy to your personal skill set, you might discover you have data you can turn into useful information. ■

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

Thanks to the following Microsoft technical experts for reviewing this article: *Dan Liebling and Kirk Olynyk*

JOIN US on the CAMPAIGN TRAIL in 2016!

LAS VEGAS 2016 CAMPAIGN FOR CODE

MARCH 7 - 11
BALLY'S HOTEL & CASINO
LAS VEGAS, NV
vslive.com/lasvegas
See pages 40 – 41 for more details

LAST CHANCE!

AUSTIN 2016 CAMPAIGN FOR CODE

MAY 16 - 19
HYATT AUSTIN, TX
vslive.com/austin
See pages 52 – 53 for more info

BOSTON 2016 CAMPAIGN FOR CODE

JUNE 13 - 16
HYATT CAMBRIDGE, MA
vslive.com/boston
See pages 54 – 55 for more info

REDMOND 2016 CAMPAIGN FOR CODE

AUGUST 8 - 12
MICROSOFT HQ, REDMOND, WA
vslive.com/redmond
See pages 70 – 71 for more info

ANAHEIM 2016 CAMPAIGN FOR CODE

SEPTEMBER 26 - 29
HYATT ORANGE COUNTY, CA – A DISNEYLAND® GOOD NEIGHBOR HOTEL
vslive.com/anaheim

WASHINGTON D.C. 2016 CAMPAIGN FOR CODE

OCTOBER 3 - 6
RENAISSANCE, WASHINGTON, D.C.
vslive.com/dc

ORLANDO 2016 CAMPAIGN FOR CODE

PART OF LIVE! 360

DECEMBER 5 - 9
LOEWS ROYAL PACIFIC
ORLANDO, FL


DETAILS COMING SOON!

AUSTIN AGENDA AT-A-GLANCE

ALM / DevOps	ASP.NET	Cloud Computing	Database and Analytics	JavaScript / HTML5 Client	Mobile Client	Software Client	UX / Design	Visual Studio / .NET	Windows Client

START TIME	END TIME	Visual Studio Live! Pre-Conference Workshops: Monday, May 16, 2016 *(Separate entry fee required)*			
7:30 AM	9:00 AM	Pre-Conference Workshop Registration • Coffee and Morning Pastries			
9:00 AM	6:00 PM	**M01** Workshop: DevOps in a Day - *Brian Randell*	**M02** Workshop: SQL Server for Developers - *Andrew Brust and Leonard Lobel*	**M03** Workshop: Building Modern Mobile Apps - *Brent Edwards and Kevin Ford*	
6:45 PM	9:00 PM	Dine-A-Round			

START TIME	END TIME	Visual Studio Live! Day 1: Tuesday, May 17, 2016			
8:00 AM	9:00 AM	**KEYNOTE:** To Be Announced, ***Tarek Madkour***, *Principal Group Program Manager, Microsoft*			
9:15 AM	10:30 AM	**T01** ASP.NET Core 1.0 in All Its Glory - *Adam Tuliper*	**T02** WCF & Web API: Can We All Just Get Along?!? - *Miguel Castro*	**T03** Using Visual Studio Tools for Apache Cordova to Create Multi-Platform Applications - *Kevin Ford*	**T04** Developer Productivity in Visual Studio 2015 - *Robert Green*
10:45 AM	12:00 PM	**T05** Build Data-Driven Web Apps with ASP.NET MVC 6 and WebAPI 2 - *Rachel Appel*	**T06** What's New in SQL Server 2016 - *Leonard Lobel*	**T07** From Oculus to HoloLens: Building Virtual & Mixed Reality Apps & Games - *Nick Landry*	**T08** This session is sequestered, details will be released soon
12:00 PM	1:30 PM	Lunch • Visit Exhibitors			
1:30 PM	2:45 PM	**T09** What You Need To Know About ASP.NET 5 and MVC 6 - *Rachel Appel*	**T10** Database Development with SQL Server Data Tools - *Leonard Lobel*	**T11** This session is sequestered, details will be released soon	**T12** Linux and OSX for Windows Developers - *Brian Randell*
3:00 PM	4:15 PM	**T13** UX Beyond the Keyboard: Gaze, Speech and Other Interfaces - *John Alexander*	**T14** Cloud Enable .NET Client LOB Applications - *Robert Green*	**T15** Go Mobile with C#, Visual Studio, and Xamarin - *James Montemagno*	**T16** Busting .NET Myths - *Jason Bock*
4:15 PM	5:30 PM	Welcome Reception			

START TIME	END TIME	Visual Studio Live! Day 2: Wednesday, May 18, 2016			
8:00 AM	9:15 AM	**W01** Angular 2 101 - *Deborah Kurata*	**W02** Building for the Internet of Things: Hardware, Sensors & the Cloud - *Nick Landry*	**W03** Creating Great Looking Android Applications Using Material Design - *Kevin Ford*	**W04** DevOps - *Brian Randell*
9:30 AM	10:45 AM	**W05** Hack Proofing Your Modern Web Applications - *Adam Tuliper*	**W06** Setting Up Your First VM, and Maybe Your Data Center, In Azure - *Eric D. Boyd*	**W07** Developing with Xamarin & Amazon AWS to Scale Native Cross-Platform Mobile Apps - *James Montemagno*	**W08** Easy and Fun Environment Creation with DevOps Provisioning Tools and Visual Studio - *John Alexander*
11:00 AM	12:00 PM	**General Session:** To Be Announced, ***Billy Hollis***, *Next Version Systems*			
12:00 PM	1:30 PM	Birds-of-a-Feather Lunch • Visit Exhibitors			
1:30 PM	2:45 PM	**W09** AngularJS & ASP.NET MVC Playing Nice - *Miguel Castro*	**W10** This session is sequestered, details will be released soon	**W11** Big Data and Hadoop with Azure HDInsight - *Andrew Brust*	**W12** Real World Scrum with Team Foundation Server 2015 & Visual Studio Team Services - *Benjamin Day*
3:00 PM	4:15 PM	**W13** Angular 2 Forms and Validation - *Deborah Kurata*	**W14** Introduction to the Next Generation of Azure PaaS—Service Fabric and Containers - Vishwas Lele	**W15** No Schema, No Problem!—Introduction to Azure DocumentDB - *Leonard Lobel*	**W16** Effective Agile Software Requirements - *Richard Hundhausen*
4:30 PM	5:45 PM	**W17** Busy JavaScript Developer's Guide to ECMAScript 6 - *Ted Neward*	**W18** Building "Full Stack" Applications with Azure App Service - *Vishwas Lele*	**W19** Predicting the Future Using Azure Machine Learning - *Eric D. Boyd*	**W20** Acceptance Testing in Visual Studio 2015 - *Richard Hundhausen*
7:00 PM	9:00 PM	Rollin' On the River Bat Cruise			

START TIME	END TIME	Visual Studio Live! Day 3: Thursday, May 19, 2016			
8:00 AM	9:15 AM	**TH01** Role-Based Security is Stinks: How to Implement Better Authorization in ASP.NET & WebAPI - *Benjamin Day*	**TH02** Code Reactions—An Introduction to Reactive Extensions - *Jason Bock*	**TH03** Busy Developer's Guide to NoSQL - *Ted Neward*	**TH04** Open Source Software for Microsoft Developers - *Rockford Lhotka*
9:30 AM	10:45 AM	**TH05** Get Good at DevOps: Feature Flag Deployments with ASP.NET, WebAPI, & JavaScript - *Benjamin Day*	**TH06** DI Why? Getting a Grip on Dependency Injection - *Jeremy Clark*	**TH07** Power BI 2.0: Analytics in the Cloud and in Excel - *Andrew Brust*	**TH08** Architects? We Don't Need No Stinkin' Architects! - *Michael Stiefel*
11:00 AM	12:15 PM	**TH09** Future of the Web with HTTP2 - *Ben Dewey*	**TH10** Unit Testing Makes Me Faster: Convincing Your Boss, Your Co-Workers, and Yourself - *Jeremy Clark*	**TH11** User Experience Case Studies—Good and Bad - *Billy Hollis*	**TH12** This session is sequestered, details will be released soon
12:15 PM	1:30 PM	Lunch			
1:30 PM	2:45 PM	**TH13** TypeScript: Work Smarter Not Harder with JavaScript - *Allen Conway*	**TH14** Exceptional Development: Dealing With Exceptions in .NET - *Jason Bock*	**TH15** Let's Write a Windows 10 App: A Basic Introduction to Universal Apps - *Billy Hollis*	**TH16** Clean Code: Homicidal Maniacs Read Code, Too! - *Jeremy Clark*
3:00 PM	4:15 PM	**TH17** Unit Testing JavaScript Code in Visual Studio .NET - *Allen Conway*	**TH18** Async Patterns for .NET Development - *Ben Dewey*	**TH19** Windows 10 Design Guideline Essentials - *Billy Hollis*	**TH20** Architecting For Failure: How to Build Cloud Applications - *Michael Stiefel*

Speakers and sessions subject to change

DETAILS COMING SOON! These sessions have been sequestered by our conference chairs. Be sure to check vslive.com/austin for session updates!

CONNECT WITH VISUAL STUDIO LIVE!

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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

VSLIVE.COM/AUSTIN

BOSTON AGENDA AT-A-GLANCE

ALM / DevOps	Cloud Computing	Database and Analytics	Mobile Client	Software Practices	UX / Design	Visual Studio / .NET Framework	Web Client	Web Server

START TIME	END TIME	Visual Studio Live! Pre-Conference Workshops: Monday, June 13, 2016 *(Separate entry fee required)*		
7:30 AM	9:00 AM	Pre-Conference Workshop Registration • Coffee and Morning Pastries		
9:00 AM	6:00 PM	**M01** Workshop: SQL Server for Developers - *Andrew Brust and Leonard Lobel*	**M02** Workshop: DevOps for Your Mobile Apps and Services - *Brian Randel*	**M03** Workshop: Native Mobile App Development for iOS, Android and Windows Using C# - *Marcel de Vries and Roy Cornelissen*
6:45 PM	9:00 PM	Dine-A-Round		

START TIME	END TIME	Visual Studio Live! Day 1: Tuesday, June 14, 2016			
8:00 AM	9:00 AM	Keynote: To Be Announced			
9:15 AM	10:30 AM	**T01** Technical Debt —Fight It with Science and Rigor - *Brian Randell*	**T02** What's New in SQL Server 2016 - *Leonard Lobel*	**T03** Getting Started with Aurelia - *Brian Noyes*	**T04** Developer Productivity in Visual Studio 2015 - *Robert Green*
10:45 AM	12:00 PM	**T05** Automated X-Browser Testing of Your Web Apps with Visual Studio CodedUI - *Marcel de Vries*	**T06** Database Lifecycle Management and the SQL Server Database - *Brian Randell*	**T07** Angular 2 101 - *Deborah Kurata*	**T08** This session is sequestered, details will be released soon
12:00 PM	1:30 PM	Lunch • Visit Exhibitors			
1:30 PM	2:45 PM	**T09** ASP.NET Core 1.0 in all its glory - *Adam Tuliper*	**T10** Predicting the Future Using Azure Machine Learning - *Eric D. Boyd*	**T11** TypeScript for C# Developers - *Chris Klug*	**T12** This session is sequestered, details will be released soon
3:00 PM	4:15 PM	**T13** Hack Proofing Your Modern Web Applications - *Adam Tuliper*	**T14** No Schema, No Problem!—Introduction to Azure DocumentDB - *Leonard Lobel*	**T15** Angular 2 Forms and Validation - *Deborah Kurata*	**T16** Windows 10—The Universal Application: One App To Rule Them All? - *Laurent Bugnion*
4:15 PM	5:30 PM	Welcome Reception			

START TIME	END TIME	Visual Studio Live! Day 2: Wednesday, June 15, 2016			
8:00 AM	9:15 AM	**W01** AngularJS & ASP.NET MVC Playing Nice - *Miguel Castro*	**W02** Cloud Enable .NET Client LOB Applications - *Robert Green*	**W03** Creating Great Windows Universal User Experiences - *Danny Warren*	**W04** Building Cross-Platform C# Apps with a Shared UI Using Xamarin.Forms - *Nick Landry*
9:30 AM	10:45 AM	**W05** Did a Dictionary and a Func Just Become the New Black in ASP.NET Development? - *Chris Klug*	**W06** Azure Mobile Apps: APIs in the Cloud for Your Mobile Needs - *Danny Warren*	**W07** User Experience Case Studies—The Good and The Bad - *Billy Hollis*	**W08** Build Cross-Platform Mobile Apps with Ionic, Angular, and Cordova - *Brian Noyes*
11:00 AM	12:00 PM	**General Session:** To Be Announced, ***Tim Huckaby***, *Founder & Chairman, InterKnowlogy & Actus Interactive Software*			
12:00 PM	1:30 PM	Birds-of-a-Feather Lunch • Visit Exhibitors			
1:30 PM	2:45 PM	**W09** Service Oriented Tech: Designing, Developing, & Implementing WCF & the Web API - *Miguel Castro*	**W10** Breaking Down Walls with Modern Identity - *Eric D. Boyd*	**W11** Knockout in 75 Minutes (Or Less...) - *Christopher Harrison*	**W12** Mobile App Development with Xamarin and F# - *Rachel Reese*
3:00 PM	4:15 PM	**W13** Securing Client JavaScript Apps - *Brian Noyes*	**W14** Exploring Microservices in a Microsoft Landscape - *Marcel de Vries*	**W15** Learning to Live Without Data Grids in Windows 10 - *Billy Hollis*	**W16** Conquer the Network—Making Your C# Mobile App More Resilient to Network Hiccups - *Roy Cornelissen*
4:30 PM	5:45 PM	**W17** Get Good at DevOps: Feature Flag Deployments with ASP.NET, WebAPI, & JavaScript - *Benjamin Day*	**W18** Patterns and Practices for Real-World Event-Driven Microservices - *Rachel Reese*	**W19** Take Your Site From Ugh to OOH with Bootstrap - *Philip Japikse*	**W20** Strike Up a Conversation with Cortana on Windows 10 - *Walt Ritscher*
7:00 PM	9:00 PM	VSLive!'s Boston By Land and Sea Tour			

START TIME	END TIME	Visual Studio Live! Day 3: Thursday, June 16, 2016			
8:00 AM	9:15 AM	**TH01** Windows Presentation Foundation (WPF) 4.6 - *Laurent Bugnion*	**TH02** Open Source Software for Microsoft Developers - *Rockford Lhotka*	**TH03** Real World Scrum with Team Foundation Server 2015 & Visual Studio Team Services - *Benjamin Day*	**TH04** Windows for Makers: Raspberry Pi, Arduino & IoT - *Nick Landry*
9:30 AM	10:45 AM	**TH05** Power BI 2.0: Analytics in the Cloud and in Excel - *Andrew Brust*	**TH06** This session is sequestered, details will be released soon	**TH07** Automate Your Builds with Visual Studio Team Services or Team Foundation Server - *Tiago Pascoal*	**TH08** Automated UI Testing for iOS and Android Mobile Apps - *James Montemagno*
11:00 AM	12:15 PM	**TH09** Big Data and Hadoop with Azure HDInsight - *Andrew Brust*	**TH10** This session is sequestered, details will be released soon	**TH11** Cross Platform Continuous Delivery with Team Build and Release Management - Tiago Pascoal	**TH12** This session is sequestered, details will be released soon
12:15 PM	1:45 PM	Lunch			
1:45 PM	3:00 PM	**TH13** Top 10 Entity Framework Features Every Developer Should Know - *Philip Japikse*	**TH14** Architecting For Failure: How to Build Cloud Applications - *Michael Stiefel*	**TH15** JavaScript Patterns for the C# Developer - *Ben Hoelting*	**TH16** Developing with Xamarin & Amazon AWS to Scale Native Cross-Platform Mobile Apps - *James Montemagno*
3:15 PM	4:30 PM	**TH17** Pretty, Yet Powerful. How Data Visualization Transforms the Way We Comprehend Information - *Walt Ritscher*	**TH18** Architects? We Don't Need No Stinkin' Architects! - *Michael Stiefel*	**TH19** Unit Testing & Test-Driven Development (TDD) for Mere Mortals - *Benjamin Day*	**TH20** Advanced Mobile App Development for the Web Developer - *Ben Hoelting*

Speakers and sessions subject to change

DETAILS COMING SOON! These sessions have been sequestered by our conference chairs. Be sure to check vslive.com/boston for session updates!

CONNECT WITH VISUAL STUDIO LIVE!

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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

VSLIVE.COM/BOSTON

JAMES MCCAFFREY

Neural Network Regression

The goal of a regression problem is to predict the value of a numeric variable (usually called the dependent variable) based on the values of one or more predictor variables (the independent variables), which can be either numeric or categorical. For example, you might want to predict the annual income of a person based on age, sex (male or female) and years of education.

The simplest form of regression is called linear regression (LR). An LR prediction equation might look like this: income = 17.53 + (5.11 * age) + (-2.02 * male) + (-1.32 * female) + (6.09 * education). Although LR is useful for some problems, in many situations it's not effective. But there are other common types of regression—polynomial regression, general linear model regression and neural network regression (NNR). Arguably, the last type of regression is the most powerful form of regression.

The most common type of neural network (NN) is one that predicts a categorical variable. For example, you might want to predict a person's political inclination (conservative, moderate, liberal) based on factors such as age, income and sex. An NN classifier has n output nodes, where n is the number of values that the dependent variable can take. The values of the n output nodes sum to 1.0 and can be loosely interpreted as probabilities. So, for predicting political inclination, an NN classifier would have three output nodes. If the output node values were (0.24, 0.61, 0.15), the NN classifier is predicting "moderate" because the middle node has the largest probability.

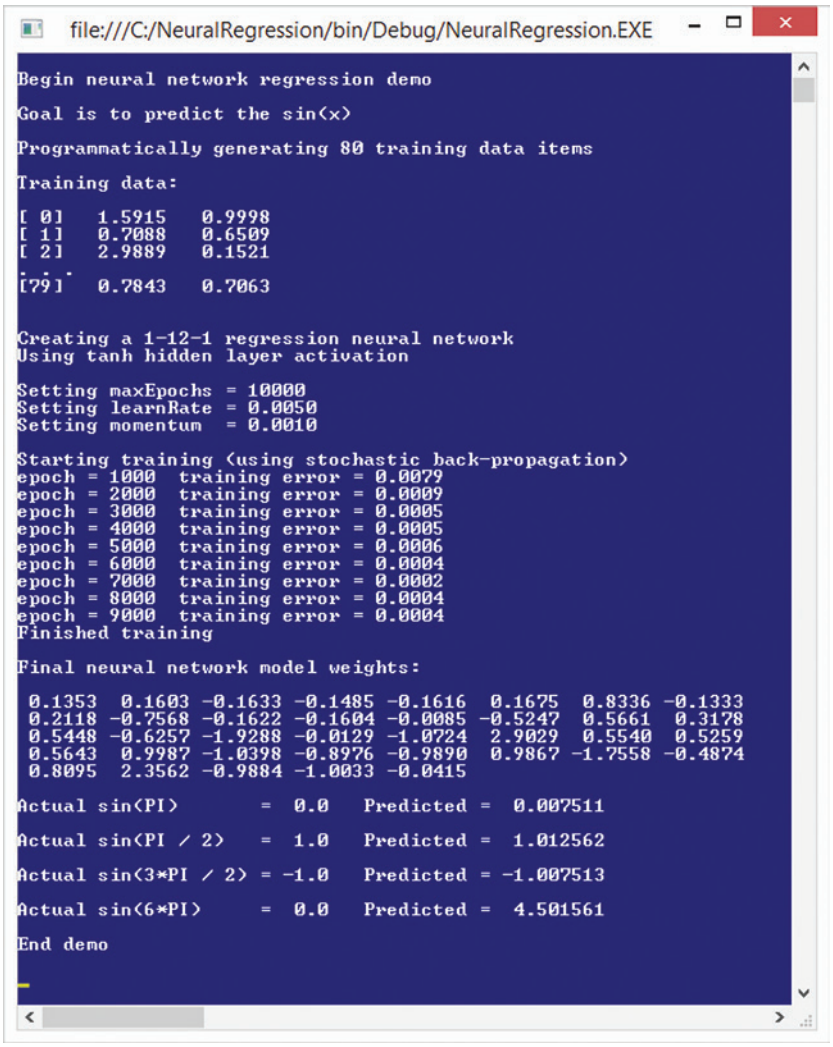


Figure 1 **Neural Network Regression Demo**

In NN regression, the NN has a single output node that holds the predicted value of the dependent numeric variable. So, for the example that predicts annual income, there would be three input nodes (one for age, one for sex where male = -1 and female = +1, and one for years of education), and one output node (annual income).

Code download available at msdn.com/magazine/0316magcode.

A good way to understand what NN regression is and to see where this article is headed, is to take a look at the demo program in **Figure 1**. Rather than tackling a realistic problem, in order to keep the ideas of NN regression as clear as possible, the goal of the demo is to create an NN model that can predict the value of the sine function. In case your trigonometry knowledge is a bit rusty, the graph of the sine function is shown in **Figure 2**. The sine function accepts a single real input value from negative infinity to positive

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

Full Integration with Microsoft Visual Studio

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

Open Source

NCache™

FREE Download

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

infinity and returns a value between -1.0 and +1.0. The sine function returns 0 when x = 0.0, x = pi (~3.14), x = 2 * pi, x= 3 * pi, and so on. The sine function is a surprisingly difficult function to model.

The demo starts by programmatically generating 80 data items to be used for training the NN model. The 80 training items have a random x input value between 0 and 6.4 (a bit more than 2 * pi) and a corresponding y value, which is the sin(x).

The demo creates a 1-12-1 NN, that is, an NN with one input node (for x), 12 hidden processing nodes (that effectively define the prediction equation), and one output node (the predicted sine of x). When working with NNs, there's always experimentation involved; the number of hidden nodes was determined by trial and error.

NN classifiers have two activation functions, one for the hidden nodes and one for the output nodes. The output node activation function for a classifier is almost always the softmax function because softmax produces values that sum to 1.0. The hidden node activation function for a classifier is usually either the logistic sigmoid function or the hyperbolic tangent function (abbreviated tanh). But in NN regression, there's a hidden node activation function, but no output node activation function. The demo NN uses the tanh function for hidden node activation.

The output of an NN is determined by its input values and a set of constants called the weights and biases. Because biases are really just special kinds of weights, the term "weights" is sometimes used to refer to both. A neural network with i input nodes, j hidden nodes, and k output nodes has a total of (i * j) + j + (j * k) + k weights and biases. So the 1-12-1 demo NN has (1 * 12) + 12 + (12 * 1) + 1 = 37 weights and biases.

The process of determining the values of the weights and the biases is called training the model. The idea is to try different values of the weights and biases to determine where the computed output values of the NN closely match the known correct output values of the training data.

There are several different algorithms that can be used to train an NN. By far the most common approach is to use the back-propagation algorithm. Back propagation is an iterative process in which values of the weights and biases slowly change, so that the NN usually computes more accurate output values.

Back propagation uses two required parameters (maximum number of iterations and learning rate) and one optional parameter (the momentum rate). The maxEpochs parameter sets a limit on the number of algorithm iterations. The learnRate parameter controls how much the weights and bias values can change in each iteration. The momentum parameter speeds up training and also helps prevent the back-propagation algorithm from getting stuck at a poor solution. The demo sets the value of maxEpochs to 10,000, the value of learnRate to 0.005, and the value of momentum to 0.001. These values were determined by trial and error.

When using the back-propagation algorithm for NN training, there are three variations that can be used. In batch back propagation, all training items are examined first and then all the weights and bias values are adjusted. In stochastic back propagation (also called online back propagation), after each training item is examined, all weights and bias values are adjusted. In mini-batch back propagation, all weights and bias values are adjusted after examining a specified fraction of the training items. The demo program uses the most common variant, stochastic back propagation.

The demo program displays a measure of error every 1,000 training epochs. Notice that the error values jump around a bit. After training completed, the demo displayed the values of the 37 weights and biases that define the NN model. The values of NN weights and biases don't have any obvious interpretation, but it's important to examine the values to check for bad results, for example, when one weight has an extremely large value and all the other weights are close to zero.

> NN classifiers have two activation functions, one for the hidden nodes and one for the output nodes.

The demo program concludes by evaluating the NN model. The NN predicted values of the sin(x) for x = pi, pi / 2, and 3 * pi / 2 are all within 0.02 of the correct values. The predicted value for sin(6 * pi) is very far away from the correct value. But this is an expected result because the NN was trained only to predict the values of sin(x) for x values between 0 and 2 * pi.

This article assumes you have at least intermediate-level programming skills, but doesn't assume you know much about neural network regression. The demo program is coded using C#, but you shouldn't have very much trouble refactoring the code to another language such as Visual Basic or Perl. The demo program is too long to present in its entirety in this article, but the complete source is available in the accompanying code download. All normal error checking was removed from the demo to keep the size of the code small and the key ideas as clear as possible.

Demo Program Structure

To create the demo program, I launched Visual Studio and selected the C# console application template from the File | New | Project menu action. I used Visual Studio 2015, but the demo has no

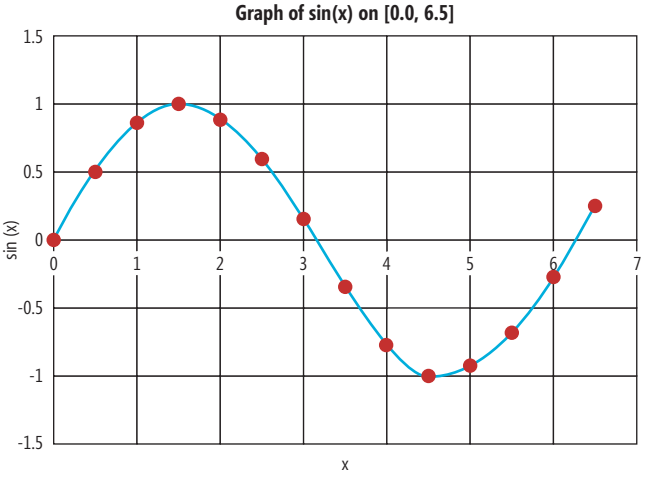


Figure 2 **The Sin(x) Function**

significant .NET dependencies so any version of Visual Studio will work. I named the project NeuralRegression.

After the template code loaded into the Editor window, in the Solution Explorer window I selected file Program.cs, right-clicked on it, and renamed it to the somewhat more descriptive NeuralRegressionProgram.cs. I allowed Visual Studio to automatically rename class Program for me. At the top of the Editor code, I deleted all reference to unused namespaces, leaving just the reference to the top-level System namespace.

The overall structure of the demo program, with a few minor edits to save space, is shown in **Figure 3**. All of the control statements are in the Main method. All of the neural network regression functionality is contained in a program-defined class named NeuralNetwork.

In the Main method, the training data is created by these statements:

```
int numItems = 80;
double[][] trainData = new double[numItems][];
Random rnd = new Random(1);

for (int i = 0; i < numItems; ++i) {
  double x = 6.4 * rnd.NextDouble();
  double sx = Math.Sin(x);
  trainData[i] = new double[] { x, sx };
}
```

As a general rule when dealing with neural networks, the more training data you have, the better. For modeling the sine function for x values between 0 and 2 * pi, I needed at least 80 items to get good results. The choice of a seed value of 1 for the random number object was arbitrary. The training data is stored in an array-of-arrays-style matrix. In realistic scenarios, you'd probably read training data from a text file.

> As a general rule when dealing with neural networks, the more training data you have, the better.

The neural network is created by these statements:

```
int numInput = 1;
int numHidden = 12;
int numOutput = 1;
int rndSeed = 0;
NeuralNetwork nn = new NeuralNetwork(numInput,
  numHidden, numOutput, rndSeed);
```

There's only one input node because the target sine function accepts only a single value. For most neural network regression problems you'll have several input nodes, one for each of the predictor-independent variables. In most neural network regression problems there's only a single output node, but it's possible to predict two or more numeric values.

An NN needs a random object to initialize weight values and to scramble the order in which training items are processed. The demo NeuralNetwork constructor accepts a seed value for the internal random object. The value used, 0, was arbitrary.

The neural network is trained by these statements:

```
int maxEpochs = 10000;
double learnRate = 0.005;
double momentum  = 0.001;
double[] weights = nn.Train(trainData, maxEpochs,
  learnRate, momentum);
ShowVector(weights, 4, 8, true);
```

An NN is extremely sensitive to the training parameter values. Even a very small change can produce a dramatically different result.

The demo program evaluates the quality of the resulting NN model by predicting the sin(x) for three standard values. The statements, with some minor edits, are:

```
double[] y = nn.ComputeOutputs(new double[] { Math.PI });
Console.WriteLine("Predicted =  " + y[0]);
y = nn.ComputeOutputs(new double[] { Math.PI / 2 });
Console.WriteLine("Predicted =  " + y[0]);
y = nn.ComputeOutputs(new double[] { 3 * Math.PI / 2.0 });
Console.WriteLine("Predicted = " + y[0]);
```

Notice that the demo NN stores its outputs in an array of output nodes, even though there's just a single output value for this

Figure 3 **Neural Network Regression Program Structure**

```
using System;
namespace NeuralRegression
{
  class NeuralRegressionProgram
  {
    static void Main(string[] args)
    {
      Console.WriteLine("Begin NN regression demo");
      Console.WriteLine("Goal is to predict sin(x)");

      // Create training data
      // Create neural network
      // Train neural network
      // Evaluate neural network

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

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

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

  }

  public class NeuralNetwork
  {
    private int numInput; // Number input nodes
    private int numHidden;
    private int numOutput;

    private double[] inputs; // Input nodes
    private double[] hiddens;
    private double[] outputs;

    private double[][] ihWeights; // Input-hidden
    private double[] hBiases;

    private double[][] hoWeights; // Hidden-output
    private double[] oBiases;

    private Random rnd;

    public NeuralNetwork(int numInput, int numHidden,
      int numOutput, int seed) { . . }

    // Misc. private helper methods

    public void SetWeights(double[] weights) { . . }
    public double[] GetWeights() { . . }

    public double[] ComputeOutputs(double[] xValues) { . . }

    public double[] Train(double[][] trainData,
      int maxEpochs, double learnRate,
      double momentum) { . . }

  } // class NeuralNetwork
} // ns
```

example. Returning an array allows you to predict multiple values without changing the source code.

The demo concludes by predicting the sin(x) for an x value that's well outside the range of the training data:

```
y = nn.ComputeOutputs(new double[] { 6 * Math.PI });
Console.WriteLine("Predicted =  " + y[0]);
Console.WriteLine("End demo");
```

In most NN classifier scenarios, you call a method that calculates the classification accuracy, that is, the number of correct predictions divided by the total number of predictions. This is possible because a categorical output value is either correct or incorrect. But when working with NN regression, there's no standard way to define accuracy. If you want to calculate some measure of accuracy, it will be problem-dependent. For example, for predicting the sin(x) you could arbitrarily define a correct prediction as one that's within 0.01 of the correct value.

Computing Output Values

Most of the key differences between an NN designed for classification and one designed for regression occur in the methods that compute output and train the model. The definition of class NeuralNetwork method ComputeOutputs begins with:

```
public double[] ComputeOutputs(double[] xValues)
{
  double[] hSums = new double[numHidden];
  double[] oSums = new double[numOutput];
...
```

The method accepts an array that holds the values of the predictor-independent variables. Local variables hSums and oSums are scratch arrays that hold preliminary (before activation) values of the hidden and output nodes. Next, the independent variable values are copied into the neural network's input nodes:

```
for (int i = 0; i < numInput; ++i)
  this.inputs[i] = xValues[i];
```

Then the preliminary values of the hidden nodes are calculated by multiplying each input value by its corresponding input-to-hidden weight, and accumulating:

```
for (int j = 0; j < numHidden; ++j)
  for (int i = 0; i < numInput; ++i)
    hSums[j] += this.inputs[i] * this.ihWeights[i][j];
```

Next, the hidden node bias values are added:

```
for (int j = 0; j < numHidden; ++j)
  hSums[j] += this.hBiases[j];
```

The values of the hidden nodes are determined by applying the hidden node activation function to each preliminary sum:

```
for (int j = 0; j < numHidden; ++j)
  this.hiddens[j] = HyperTan(hSums[j]);
```

Next, the preliminary values of the output nodes are calculated by multiplying each hidden node value by its corresponding hidden-to-output weight, and accumulating:

```
for (int k = 0; k < numOutput; ++k)
  for (int j = 0; j < numHidden; ++j)
    oSums[k] += hiddens[j] * hoWeights[j][k];
```

Then the hidden node bias values are added:

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

Up to this point, computing output node values for a regression network is exactly the same as computing output node values for a classifier network. But in a classifier, the final output node values would be computed by applying the softmax activation function to each accumulated sum. For a regression network no activation function is applied. Therefore, method ComputeOutputs concludes by simply copying the values in the oSums scratch array directly to the output nodes:

```
...
  Array.Copy(oSums, this.outputs, outputs.Length);
  double[] retResult = new double[numOutput]; // Could define a GetOutputs
  Array.Copy(this.outputs, retResult, retResult.Length);
  return retResult;
}
```

For convenience, the values in the output nodes are also copied to a local return array so they can be easily accessed without calling a GetOutputs method of some sort.

When training an NN classifier using the back-propagation algorithm, the calculus derivatives of the two activation functions are used. For the hidden nodes the code looks like:

```
for (int j = 0; j < numHidden; ++j) {
  double sum = 0.0; // sums of output signals
  for (int k = 0; k < numOutput; ++k)
    sum += oSignals[k] * hoWeights[j][k];
  double derivative = (1 + hiddens[j]) * (1 - hiddens[j]);
  hSignals[j] = sum * derivative;
}
```

The value for the local variable named derivative is the calculus derivative of the tanh function and comes from quite complex theory. In an NN classifier, the calculation involving the derivative of the output node activation function is:

```
for (int k = 0; k < numOutput; ++k) {
  double derivative = (1 - outputs[k]) * outputs[k];
  oSignals[k] = (tValues[k] - outputs[k]) * derivative;
}
```

Here, the value for local variable derivative is the calculus derivative of the softmax function. However, because NN regression doesn't use an activation function for output nodes, the code is:

```
for (int k = 0; k < numOutput; ++k) {
  double derivative = 1.0;
  oSignals[k] = (tValues[k] - outputs[k]) * derivative;
}
```

Of course, multiplying by 1.0 has no effect so you could simply drop the derivative term. Another way of thinking about this is that in NN regression, the output node activation function is the identity function f(x) = x. The calculus derivative of the identity function is the constant 1.0.

Wrapping Up

The demo code and explanation in this article should be enough to get you up and running if you want to explore neural network regression with one or more numeric predictor variables. If you have a predictor variable that's categorical, you'll need to encode the variable. For a categorical predictor variable that can take one of two possible values, such as sex (male, female), you'd encode one value as -1 and the other as +1.

For a categorical predictor variable that can take three or more possible values, you'd use what's called 1-of-(N-1) encoding. For example, if a predictor variable is color that can take one of four possible values (red, blue, green, yellow), then red would be encoded as (1, 0, 0), blue as (0, 1, 0), green as (0, 0, 1), and yellow as (-1, -1, -1).■

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

Thanks to the following Microsoft technical experts who reviewed this article: *Gaz Iqbal and Umesh Madan*

Spreadsheets Made Easy.

Fastest Calculations

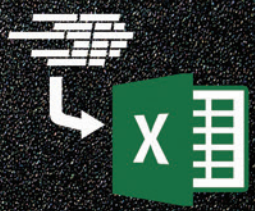
Evaluate complex Excel-based models and business rules with the fastest and most complete Excel-compatible calculation engine available.

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

TED NEWARD

How To Be MEAN: Robust Validation with MongooseJS

In my February 2016 column (msdn.com/magazine/mt632276), I transitioned to a MongoDB database. Doing so wasn't that hard thanks to the JSON-based nature of Node.js and the JSON-based nature of MongoDB. (Life is always easier when working with data if the transition is from apples to apples). MongoDB has a great advantage in that it will "scale up" and "scale out" easily, not to mention that it's trivial to get started. But it also has a significant drawback: Because MongoDB is a "schemaless" database (in that the schema is already predefined—a database holds collections, collections hold documents and documents basically are just JSON objects), within it lies the seeds of its own destruction.

> By now, the exercise should be getting pretty simple, straightforward and repetitive: "Which 'thing' do you npm this time?"

First, recall that MongoDB queries are essentially query documents (that first parameter to the find call), containing the fields by which to scan the collection for matches. So if the query "{'fristName': 'Ted'}" gets executed against the "persons" collection in the existing database, nothing will come back—the field name in the query document is misspelled ("fristName" instead of "firstName"), so it will match against no documents in the collection. (Unless there's a typo in the document in the collection, of course.) This is one of the biggest disadvantages of a schemaless database—a simple typo in the code or user input can create accidental bugs of the most furious head-scratching nature. This is where it would be nice to get some language support, whether that's by a compiler or an interpreter.

Second, notice that the code displayed in my last column is reminiscent of the "two-tier" applications that were popular for two decades during the "client-server" era of computing. There's a code layer that takes input directly from the user (or, in this case, from the API consumer), applies it and gets a simple data structure back from the database, which it hands directly back to the caller. There's certainly no sense of "object-orientation" in that code. While not a deal-breaker, it would be nice if we could get a stronger sense of "object-ness" to the server-side code, such that some validation of various properties could be centralized in one place, for example. Case in point: Is it legal for a person to have an empty firstName or lastName? Can he be doing absolutely anything in his status? Is there a "default" status, if he chooses not to provide one, or is an empty status acceptable?

The Node.js crowd has wrestled with these problems for quite a while (it was one of the first language ecosystems to adopt MongoDB on a large-scale basis) and, not surprisingly, it has come up with an elegant solution to the problem, called MongooseJS. It's a software layer that sits "on top" of MongoDB and provides not only a schema-like language-verified validation layer, but also an opportunity to build a layer of "domain object" into the server-side code. Hence, it's sort of "the other 'M'" in the MEAN stack.

MongooseJS: Getting Started

By now, the exercise should be getting pretty simple, straightforward and repetitive: "Which 'thing' do you npm this time?" The short answer is "npm install --save mongoose," but if there's ever some confusion as to what the exact package might be (the Node folks tend to waffle between "thing" and "thingjs" as package names), an "npm find" or "npm search" will search the npm registry for packages that match whatever terms follow on the command line. Alternatively, typing "Mongoose JS" into the search engine of choice will yield up the Mongoose Web site (mongoosejs.com), which will have the correct npm incarnation, along with a ton of documentation on how to use Mongoose. (This makes it a handy thing to have bookmarked in the browser, for sure.)

Once installed, you can begin to define Mongoose "schema" objects, which will define the kind of objects to be stored in the MongoDB collection.

Figure 1 **Rewriting the getAllPersons Route**

```
var getAllPersons = function(req, res) {
  Person.find(function(err, persons) {
    if (err) {
     debug("getAllPersons--ERROR:",err);
      res.status(500).jsonp(err);
    }
    else {
      debug("getAllPersons:", persons);
      res.status(200).jsonp(persons);
    }
});
};
```

Figure 2 **Rewriting the personId Route**

```
app.param('personId', function (req, res, next, personId) {
  debug("personId found:",personId);
  if (mongodb.ObjectId.isValid(personId)) {
    Person.findById(personId)
      .then(function(person) {
        debug("Found", person.lastName);
        req.person = person;
        next();
      });
  }
  else {
    res.status(404).jsonp({ message: 'ID ' + personId + ' not found'});
  }
});
```

Mongoosing a Person

Mongoose uses some interesting terminology for what's essentially a two-step process to defining a JavaScript object model on top of the MongoDB database API. First, we define a "schema," which looks like a traditional class from a more traditional class-based language (C#, C++, Java or Visual Basic). This schema will have fields, define types for these fields and optionally include some validation rules around the fields for when assigning values to the fields. You can also add some methods, instance or static, which I'll get to later. Then, once the schema object is defined, you "compile" it into a Model, which is what will be used to construct instances of these objects.

> Having defined the model, now the only thing left to do is to rewrite the various route methods that will use the model.

Take careful note here: This is still JavaScript, and more important, this is the version of JavaScript more accurately described as ECMAScript 5, which lacks any concept of "class" whatsoever. Therefore, even though the Mongoose approach creates the illusion that you're defining a "class," it's still the prototype-based language that you know and love (or loathe). This is partly the reason for this two-step process—the first to define an object that will serve as the class, the second to define an object that will implicitly work as the "constructor" or "factory," subject to the JavaScript/ECMAScript 5 rules around the "new" operator.

So, translating that into code, after "require()"ing the Mongoose library into the usual "mongoose" local variable at the top of the JavaScript code, you can use that to define a new Schema object:

```
// Define our Mongoose Schema
var personSchema = mongoose.Schema({
  firstName: String,
  lastName: String,
  status: String
});
var Person = mongoose.model('Person', personSchema);
```

There's a variety of things you can do with the fields in the personSchema, but for starters, let's keep it simple; this will not only help get to working code faster, but will also highlight a few niceties about Mongoose along the way. Also, again, notice the two-step process by which the model is defined: first, you define the schema, using the Schema function/constructor, and then pass the schema object into the model function/constructor, along with the name of this model. Convention among Mongoose users is that the returned object from the model call will be the same as the type being defined because that's what will help give it the illusion of being a class.

Mongoosing in Action

Having defined the model, now the only thing left to do is to rewrite the various route methods that will use the model. The easiest place to start is the getAllPersons route, as shown in **Figure 1**, because it returns the entire collection from the database, with no filters.

Notice how the code is fundamentally similar to what was there before—the query is executed and invokes the passed-in callback function (again, with the convention "err, result" as its parameters) when the query is complete. However, now Mongoose provides a tiny bit more structure around the access by routing it through the Personmodel object, which will yield all kinds of powerful benefits, as you'll see in a minute.

Next up is the personId middleware, because it's used in almost all the rest of the routes, as shown in **Figure 2**.

Again, this is a middleware function, so the goal is to find the Person object out of the collection and store it into the request object (req). But notice how after validating that the incoming personId is a valid MongoDB ObjectId/OID; the Person object shows up again, this time calling findById, passing in the ObjectID/OID passed in. Of most interest here is that Mongoose supports

Figure 3 **Instantiating a New Person Model**

```
var updatePerson = function(req, res) {
  debug("Updating",req.person,"with",req.body);

  _.merge(req.person, req.body);
  // The req.person is already a Person, so just update()
  req.person.save(function (err, person) {
    if (err)
      res.status(500).jsonp(err);
    else {
      res.status(200).jsonp(person);
    }
  });
};
var insertPerson = function(req, res) {
  var person = new Person(req.body);
  debug("Received", person);
  person.save(function(err, person) {
    if (err)
      res.status(500).jsonp(err);
    else
      res.status(200).jsonp(person);
  });
};
var deletePerson = function(req, res) {
  debug("Removing", req.person.firstName, req.person.lastName);
  req.person.delete(function(err, result) {
    if (err) {
      debug("deletePerson: ERROR:", err);
      res.status(500).jsonp(err);
    }
    else {
      res.status(200).jsonp(req.person);
    }
  });
};
```

Figure 4 **Mongoose Validation**

```
// Define our Mongoose Schema
var personSchema = mongoose.Schema({
  firstName: {
    type: String,
    required: true,
    default: "(No name specified)"
  },
  lastName: {
    type: String,
    required: true,
    default: "(No name specified)"
  },
  status: {
    type: String,
    required: true,
    enum: [
      "Reading MSDN",
      "WCFing",
      "RESTing",
      "VBing",
      "C#ing"
    ],
    default: "Reading MSDN"
  },
});
var Person = mongoose.model('Person', personSchema);
```

the "promise" syntax/style that will be a fixture in future versions of JavaScript—the returned object from findById is a Promise object, upon which you can invoke "then" and pass in a callback to be executed when the query is finished configuring.

This two-step approach then lets you do a whole host of things with this query before its execution, such as sort the query results in a particular order based on fields within the results, or select only a subset of the fields (so as to hide any sensitive information the client shouldn't receive). These would be method calls wedged right in between find and then in a fluid style, such as:

```
Person.find({ })
  .sort({ 'firstName':'asc', 'lastName':'desc' })
  .select('firstName lastName status')
  .then(function(persons) {
    // Do something with the returned persons
  });
```

Figure 5 **Error Result from Failing Validation on a Save**

```
{
  "message":"Person validation failed",
  "name":"ValidationError",
  "errors":{
    "status":{
      "properties":{
        "enumValues":[
          "Reading MSDN",
          "WCFing",
          "RESTing",
          "VBing",
          "C#ing"
        ],
        "type":"enum",
        "message":"`{VALUE}` is not a valid enum value for path `{PATH}`.",
        "path":"status",
        "value":"Javaing"
      },
      "message":"`Javaing` is not a valid enum value for path `status`.",
      "name":"ValidatorError",
      "kind":"enum",
      "path":"status",
      "value":"Javaing"
    }
  }
}
```

In this case, this would sort the results by firstName in ascending order, sort the results by lastName in descending order (not that doing that makes much sense) and then strip out anything except the "firstName," "lastName" and "status" fields. The full list of query "modifiers" is impressive and includes a number of useful methods such as limit (cut off at a certain number of results), skip (to skip past the first *n* results returned), count (to return an aggregate count of the documents returned) and so on.

Before you get too distracted by that, however, I'll round out the rest of the route methods.

Having used the middleware to obtain the Person object in question for update and delete, those become pretty straightforward uses of the save and delete methods provided by Mongoose on the objects themselves. As shown in **Figure 3**, inserting a new Person just requires instantiating a new Person model and using the save method on it.

> Again, this is a middleware function, so the goal is to find the Person object out of the collection and store it into the request object (req).

Conceptually, it looks a lot like the previous, non-Mongoose version, but there's a really important, if subtle, change: logic about Person is now being more localized to the Person model (and Mongoose's own persistence implementation).

Mongoose Validation

Now, just for grins, you want to add several new rules to the application: neither firstName nor lastName can be empty and status can only be one of several possible values (a la an enumerated type). This is where Mongoose's ability to create a domain object model within the server side can be particularly powerful, because classic O-O thinking holds that these kinds of rules should be encapsulated within the object type itself and not the surrounding usage code.

Figure 6 **Hooking Lifecycle Methods with Mongoose**

```
// Define our Mongoose Schema
var personSchema = mongoose.Schema({
  created: {
    type: Date,
    default: Date.now
  },
  updated: {
    type: Date,
  },
  // ... as before
});
personSchema.pre('save', function(next) {
  // Make sure updated holds the current date/time
  this.updated = new Date();
  next();
});
var Person = mongoose.model('Person', personSchema);
```

With Mongoose, this is actually somewhat trivial. Within the schema object, the fields go from being simple name: type pairs to more complex name: object descriptor pairs and validation rules can be specified in these object descriptors. In addition to the usual raft of numeric validation (min and max) and String validation (min length and max length), you can specify an array of acceptable values for the "status" field and define a default value for each field when none is specified, which (not surprisingly) neatly fits the new requirements, as shown in **Figure 4**.

Nothing else needs to change anywhere else in the code base—all of these rules about Person-ness are captured exactly where they should be, in the definition of the schema object.

If you try to now insert JSON, that doesn't obey the right list of statuses, such as:

```
{
  "firstName":"Ted",
  "lastName":"Neward",
  "status":"Javaing"
}
```

The save method called inside of the insertPerson route will yield a 500 response, with the returned JSON body reading as shown in **Figure 5**.

That pretty much nails what went wrong there.

> The key thing to realize is that this pairing of Mongoose + MongoDB provides a powerful combination.

Mongoose Methoding

Of course, object-orientation means combining state and behavior, which means that these domain objects won't really be objects unless you can attach methods to either object instances or the "class" as a whole. Doing so is actually fairly simple: You call a Mongoose method (either method for a traditional instance-based method or static for something class-based) that will define the method, and pass in the function to use as the method body, like so:

```
personSchema.method('speak', function() {
  console.log("Don't bother me, I'm", status);
});
```

It's not exactly like writing a class in C#, but it's pretty close. Note that along with all the other changes to the Schema objects, these must all be done before the Schema is compiled into the model object, so this call has to appear before the mongoose.model call.

Mongoose Versioning

By the way, if you do a quick GET to /persons again, the resulting output contains something a little unexpected:

```
[{"_id":"5681d8bfddb73cd9ff445ec2",
  "__v":0,
  "status":"RESTing",
  "lastName":"Castro",
  "firstName":"Miguel"}]
```

The "__v" field, which Mongoose silently slipped into the mix (and preserves all the way to the database) is a versioning field, known within Mongoose as the versionKey field, and it helps Mongoose recognize changes to the document; think of it as a version number on a source code file, as a way of detecting simultaneous changes. Normally, this is just an internal detail of Mongoose, but it can be a little surprising to see it show up there the first time.

Nevertheless, that raises a more interesting question: It's not uncommon for systems to want to track when a document was created or when changes were made to a document, and it would certainly be a pain to have to fill in fields every time any part of the surrounding code touched or saved one of these guys.

Mongoose can help with that, by letting you "hook" certain lifecycle methods to update fields right before the document is written to MongoDB, regardless of the call that's persisting it, as shown in **Figure 6**.

Now, whenever a Person object is constructed, it will default the created field to hold the current date/time and the updated field will be set to the current date/time just prior to being sent to MongoDB for storage. Mongoose calls these "middleware," because they are in spirit to the middleware functions defined by Express, but make no mistake, these are specific and contained entirely to the Mongoose-defined object.

Now, whenever a Person is created or updated, it'll have those corresponding fields filled in and updated as necessary. Moreover, should something more sophisticated (like a full-blown audit log) be necessary, it's fairly easy to see how this could be added. Instead of created/updated fields, there's an auditLog field, which would be an array, and the "save" hook would simply append over and over to that array to describe what was done each time and/or who did it, depending on requirements.

Wrapping Up

This has been another relatively heavy piece, and certainly there's a lot more of Mongoose that's worth exploring. However, that's half the fun of exploring a new technology stack and platform, and it would be extremely churlish of me not to include that kind of fun to my loyal readership. The key thing to realize is that this pairing of Mongoose + MongoDB provides a powerful combination: the schemaless nature of the MongoDB database is coupled with language-enforced validation of basic types, plus runtime-enforced validation of data values to give us something of the best of both the statically typed and dynamically typed worlds. It's not without problems and rough spots, but overall, it's hard for me to imagine doing any serious coding work in MongoDB without something like Mongoose keeping me from doing stupid things.

I'm almost done with the server-side of things, but I'm out of space, so for now … happy coding! ■

Ted Neward *is a Seattle-based polytechnology consultant, speaker and mentor. He has written more than 100 articles, is an F# MVP, INETA speaker, and has authored and coauthored a dozen books. Reach him at ted@tedneward.com if you're interested in having him come work with your team, or read his blog at blogs.tedneward.com.*

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

FRANK LA VIGNE

Parsing CSV Files in UWP Apps

Parsing a Comma Separated Value (CSV) file sounds easy enough at first. Quickly, however, the task becomes more and more intricate as the pain points of CSV files become clear. If you're not familiar with the format, CSV files store data in plain text. Each line in the file constitutes a record. Each record has its fields typically delineated by a comma, hence the name.

Developers today enjoy standards among data exchange formats. The CSV file "format" harkens to an earlier time in the software industry before JSON, before XML. While there's a Request for Comments (RFC) for CSV files (bit.ly/1NsQlvw), it doesn't enjoy official status. Additionally, it was created in 2005, decades after CSV files started to appear in the 1970s. As a result, there exists quite a bit of variation to CSV files and the rules are a bit murky. For instance, a CSV file might have fields separated by tabs, a semicolon or any character.

In practical terms, the Excel implementation of CSV import and export has become the de-facto standard and is seen most widely in the industry, even outside the Microsoft ecosystem. Accordingly, the assumptions I make in this article about what constitutes "correct" parsing and formatting will be based upon how Excel imports/exports CSV files. While most CSV files will fall in line with the Excel implementation, not every file will. Toward the end of this column, I introduce a strategy to handle such uncertainty.

A fair question to ask is, "Why even write a parser for a decades-old quasi-format in a very new platform?" The answer is simple: Many organizations have legacy data systems. Thanks to the file format's long life span, nearly all of these legacy data systems can export to CSV. Furthermore, it costs very little in terms of time and effort to export data to CSV. Accordingly, there are plenty of CSV-formatted files in larger enterprise and government data sets.

Designing an All-Purpose CSV Parser

Despite the lack of an official standard, CSV files typically share some common traits.

Generally speaking, CSV files: are plain text, contain one record per line, have records in each line separated by a delimiter, have one-character delimiters and present fields in the same order.

These common traits outline a general algorithm, which would consist of three steps:

1. Split a string along the line delimiter.
2. Split each line along the field delimiter.
3. Assign each field value to a variable.

This would be fairly easy to implement. The code in **Figure 1** parses the CSV input string into a List<Dictionary<string, string>>.

Code download available at bit.ly/1To1IVI.

Figure 1 **Parsing the CSV Input String into List<Dictionary<string,string>>**

```
var parsedResult = new List<Dictionary<string, string>>();
var records = RawText.Split(this.LineDelimiter);

foreach (var record in records)
  {
    var fields = record.Split(this.Delimiter);
    var recordItem = new Dictionary<string, string>();
    var i = 0;

    foreach (var field in fields)
    {
      recordItem.Add(i.ToString(), field);
      i++;
    }
    parsedResult.Add(recordItem);
  }
```

This approach works great using an example like the following office divisions and their sales numbers:

```
East, 73, 8300
South, 42, 3000
West, 35, 4250
Mid-West, 18, 1200
```

To retrieve values from the string, you would iterate through the List and pull out values in the Dictionary using the zero-based field index. Retrieving the office division field, for example, would be as simple as this:

```
foreach (var record in parsedData)
{
  string fieldOffice = record["0"];
}
```

While this works, the code isn't as readable as it could be.

A Better Dictionary

Many CSV files include a header row for the field name. The parser would be easier for developers to consume if it used the field name as a key for the dictionary. As any given CSV file might not have a header row, you should add a property to convey this information:

```
public bool HasHeaderRow { get; set; }
```

For instance, a sample CSV file with a header row might look something like this:

```
Office Division, Employees, Unit Sales
East, 73, 8300
South, 42, 3000
West, 35, 4250
Mid-West, 18, 1200
```

Ideally, the CSV parser would be able to take advantage of this piece of metadata. This would make the code more readable. Retrieving the office division field would look something like this:

```
foreach (var record in parsedData)
{
  string fieldOffice = record["Office Division"];
}
```

Blank Fields

Blank fields occur commonly in data sets. In CSV files, a blank field is represented by having an empty field in a record. The delimiter is still required. For example, if there were no Employee data for the East office, the record would look like this:

```
East,,8300
```

If there were no Unit Sales data, as well as no Employee data, the record would look like this:

```
East,,
```

Every organization has its own data-quality standards. Some may choose to place a default value in a blank field to make the CSV file more human-readable. Default values typically would be 0 or NULL for numbers and "" or NULL for strings.

Staying Flexible

Given all the ambiguities surrounding the CSV file format, the code can't make any assumptions. There's no guarantee that field delimiter will be a comma and there's no guarantee that the record delimiter will be a new line.

Accordingly, both will be properties of the CSVParser class:

```
public char Delimiter { get; set; }
public char LineDelimiter { get; set; }
```

To make it easier for developers consuming this component, you want to make default settings that will apply in most cases:

```
private const char DEFAULT_DELIMITER = ',';
private const char DEFAULT_LINE_DELIMITER = '\n';
```

Should someone wish to change the default delimiter to a tab character, the code is quite simple:

```
CsvParser csvParser = new CsvParser();
csvParser.Delimiter = '\t';
```

Escaped Characters

What would happen if the field itself contains the delimiter character, like a comma? For example, instead of referring to sales by region, what if the data had city and state? Typically, CSV files work around this by encasing the entire field in quotes, like so:

```
Office Division, Employees, Unit Sales
"New York, NY", 73, 8300
"Richmond, VA", 42, 3000
"San Jose, CA", 35, 4250
"Chicago, IL", 18, 1200
```

This algorithm would turn the one field value "New York, NY" into two discrete fields with values split along the comma, "New York" and "NY."

Figure 2 **Original Data with Quotes**

Office Division	Employees	Unit Sales	Office Motto
New York, NY	73	8300	"We sell great products"
Richmond, VA	42	3000	"Try it and you'll want to buy it"
San Jose, CA	35	4250	"Powering Silicon Valley!"
Chicago, IL	18	1200	"Great products at great value"

Figure 3 **Quote Data**

Quote
"The only thing we have to fear is fear itself." -President Roosevelt
"Logic will get you from A to B. Imagination will take you everywhere." -Albert Einstein

In this case, separating the values of city and state might not be detrimental, but there are still extra quote characters polluting the data. While they're easy enough to remove here, more complex data might not be so easy to clean up.

Now It Gets Complicated

This method of escaping commas inside fields introduces another character to be escaped: the quote character. What if, for example, there were quotes in the original data as shown in **Figure 2**?

The raw text in the CSV file itself would look like this:

```
Office Division, Employees, Unit Sales, Office Motto
"New York, NY",73,8300,"""We sell great products"""
"Richmond, VA",42,3000,"""Try it and you'll want to buy it"""
"San Jose, CA",35,4250,"""Powering Silicon Valley!"""
"Chicago, IL",18,1200,"""Great products at great value"""
```

The one quotation mark (") gets escaped into three quotation marks (""""), which adds an interesting twist to the algorithm. Of course, the first reasonable question to ask is this: Why did one quote turn into three? Just as in Office Division field, the contents of the field get surrounded by quotes. In order to escape quote characters that are part of the content, they're doubled up. Therefore, " becomes "".

Another example (**Figure 3**) might demonstrate the process more clearly.

The data in **Figure 3** would be represented in CSV as this:

```
Quote
"""The only thing we have to fear is fear itself."" -President Roosevelt"
"""Logic will get you from A to B. Imagination will take you everywhere.""
-Albert Einstein"
```

It might be clearer now that the field is wrapped in quotation marks and that the individual quotation marks in the field's content are doubled up.

Edge Cases

As I mentioned in the opening section, not all files will adhere to the Excel implementation of CSV. The lack of a true specification for CSV makes it difficult to write one parser to handle every CSV file in existence. Edge cases will most certainly exist and that means the code has to leave a door open to interpretation and customization.

Inversion of Control to the Rescue

Given the CSV format's hazy standard, it's not practical to write a comprehensive parser for all imaginable cases. It might be more ideal to write a parser to suit a particular need of an app. Using Inversion of Control lets you customize a parsing engine for a particular need.

To accomplish this, I'll create an interface to outline the two core functions of parsing: extracting records and extracting fields. I decided to make the IParserEngine interface asynchronous. This makes sure any app using this component will remain responsive no matter how large the CSV file is:

```
public interface IParserEngine
{
  IAsyncOperation<IList<string>> ExtractRecords(char lineDelimiter, string csvText);
  IAsyncOperation<IList<string>> ExtractFields(char delimiter, char quote,
    string csvLine);
}
```

Then I add the following property to the CSVParser class:

```
public IParserEngine ParserEngine { get; private set; }
```

I then offer developers a choice: use the default parser or inject their own. To make it simple, I'll overload the constructor:

```
public CsvParser()
{
  InitializeFields();
  this.ParserEngine = new ParserEngines.DefaultParserEngine();
}

public CsvParser(IParserEngine parserEngine)
{
  InitializeFields();
  this.ParserEngine = parserEngine;
}
```

The CSVParser class now provides the basic infrastructure, but the actual parsing logic is contained within the IParserEngine interface. For convenience of developers, I created the DefaultParserEngine, which can process most CSV files. I took into account the most likely scenarios developers will encounter.

Reader Challenge

I have taken into account the bulk of scenarios developers will encounter with CSV files. However, the indefinite nature of the CSV format makes creating a universal parser for all cases impractical. Factoring all the variations and edge cases would add significant cost and complexity of the cost along with impacting performance.

I'm certain that there are CSV files out "in the wild" that the DefaultParserEngine will not be able to handle. This is what makes the dependency injection pattern a great fit. If developers have a need for a parser that can handle an extreme edge case or write something more performant, they certainly are welcome to do so. Parser engines could be swapped out with no changes to the consuming code.

The code for this project is available at bit.ly/1To1IVI.

Wrapping Up

CSV files are a leftover from days gone by and, despite the best efforts of XML and JSON, are still a commonly used data exchange format. CSV files lack a common specification or standard and, while they often have common traits, are not certain to be in place in any given file. This makes parsing a CSV file a non-trivial exercise.

Given a choice, most developers would probably exclude CSV files from their solutions. However, their widespread presence in legacy enterprise and government data sets may preclude that as an option in many scenarios.

Simply put, there is a need for a CSV parser for Universal Windows Platform (UWP) apps and a real-world CSV parser has to be flexible and robust. Along the way, I demonstrated here a practical use for dependency injection to provide that flexibility. While this column and its associated code target UWP apps, the concept and code apply to other platforms capable of running C#, such as Microsoft Azure or Windows desktop development. ■

Frank La Vigne *is a technology evangelist on the Microsoft Technology and Civic Engagement team, where he helps users leverage technology in order to create a better community. He blogs regularly at FranksWorld.com and has a YouTube channel called Frank's World TV (youtube.com/FranksWorldTV).*

Thanks to the following technical expert for reviewing this article: *Rachel Appel*

REDMOND 2016 CAMPAIGN FOR CODE

VISUAL STUDIO LIVE! is on a Campaign for Code in 2016, and we're taking a swing through our home state, rallying @ **Microsoft Headquarters** in beautiful Redmond, WA. From August 8 – 12, developers, software architects, engineers, designers and more will convene for five days of unbiased and cutting-edge education on the Microsoft Platform. Plus, you'll be at the heart of it all: eating lunch with the Blue Badges, rubbing elbows with Microsoft insiders, exploring the campus, all while expanding your development skills and the ability to create better apps!

SUPPORTED BY

Microsoft | Visual Studio | msdn magazine | Visual Studio MAGAZINE

PRODUCED BY

1105 MEDIA — YOUR GROWTH. OUR BUSINESS.

MICROSOFT HQ • AUGUST 8-12, 2016

MICROSOFT HEADQUARTERS • REDMOND, WA

CONNECT WITH VISUAL STUDIO LIVE!

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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

TOPICS INCLUDE:

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

Have your code heard: register to join us today.

REGISTER NOW AND SAVE $400!

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

USE PROMO CODE VSLRED2

VSLIVE.COM/REDMOND

DAVID S. PLATT

The Internet of Invisible Things

The "Internet of Things" (IoT) is today's latest buzz word. Naturally, Microsoft wants to be a player in it. To succeed, Microsoft will have to adjust its current strategy.

Microsoft's Bryan Roper gave a presentation at the recent Consumer Electronics Show, which you can see at bit.ly/1QhDzQe. He describes his horror upon opening his IoT-connected clothes washer and finding a wasteful load of only two socks and a shirt.

"I can ask a really cool question of Cortana like this: 'Hey, Cortana, show me washloads alongside family members at home.'" His PC shows a graph of wash capacity per family member, from which he deduces that his son Billy is the water-wasting culprit.

"Not me!"

Cortana could one day help parents smoke out the lame excuse-making of their children.

I admire the guy's energy, but he's completely barking up the wrong tree. Bryan, your customer is not you. You enjoy whipping out your technology stack to discover who ran the short washload, but no one else in the universe would, because it's *easier* to look at the clothes in the washer. Shirts especially tend to identify their wearer: "Whose is this with the rat's hindquarters on it?" "Oh, that's Daddy's. It signifies what he doesn't give."

Microsoft is missing the point, demonstrating geeky things to geeks. It's the *Consumer* Electronics Show, not the *Ultra-Geek Hobbyist* Electronics Show. Microsoft needs to think like users. That means using technology to make simple things simpler, not making complex things possible.

Someone at Microsoft does actually get it: Harry Shum, director of Microsoft Research. At the Microsoft Ignite conference last summer, Shum rephrased author Arthur C. Clarke when he said, "Any sufficiently advanced technology is invisible."

For example: you're baking a cake, and use up the last of your chocolate. You simply say, "Hey, Cortana, remember we need more baking chocolate." Cortana answers back: "OK, I've got that." She'll add it to your family's shared grocery list. Whoever next goes to the supermarket will say, "Hey, Cortana, what do I need today?" and find it. Or if you get automatic deliveries, she'll add it to the next order, and it'll just show up. Her internal gyrations are completely invisible.

If you're using Cortana as Big Sister to monitor your kids' behavior, you don't think, "Cortana, show me this collection of data sorted by that index." You want to speak in terms of everyday life, such as something like: "Hey, Cortana, who keeps drinking all the milk and putting back the empty carton?" It's Cortana's job to parse that statement and answer it from whatever data she has available. "It's that little twerp Billy again. Caught him red-handed. I've cut an hour off his permitted Xbox time." Cortana might actually stop the depredations of the ghost characters, Not Me and Ida Know, who wreak havoc in Bill Keane's *Family Circus* comic series.

I can see Cortana doing scut work so we no longer have to. Imagine an IoT cat door with an RFID reader that responds to my cat's implanted ID chip. I won't have to get out of my chair when I hear meowing. "Cortana, let the cat in." "Cortana, let the cat out." "Cortana, let the damn cat in again." I'd eventually place a standing order: "Cortana, let the cat in or out when she wants to." If Cortana was really smart, she'd say something like, "Simba has a vet appointment on Tuesday at four. I'll keep her inside after midnight Monday, OK?"

The per-incident savings aren't huge, but they add up over time. My grandchildren won't believe that I actually had to find paper and pencil, write down my needed groceries; then find that particular list and remember to take it to the supermarket—just as my daughters can't imagine how in my youth I had to get up off the couch to change TV channels or adjust the volume.

I'll take Harry Shum's idea one step further: Any sufficiently advanced technology *makes other things invisible*. "Cortana, mow the lawn, but be careful of the new petunias." "Cortana, shovel the snow off my steps." I can't wait.

Here's an idea: If Microsoft wants to build a better IoT mousetrap, teach Cortana to reset all the clocks in my house after a power blip, without my needing to touch anything. *That's* the cool thing that'll make consumers, not geeks, open their wallets. ■

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.

1996
15+ years on the market

300,000+ developer community

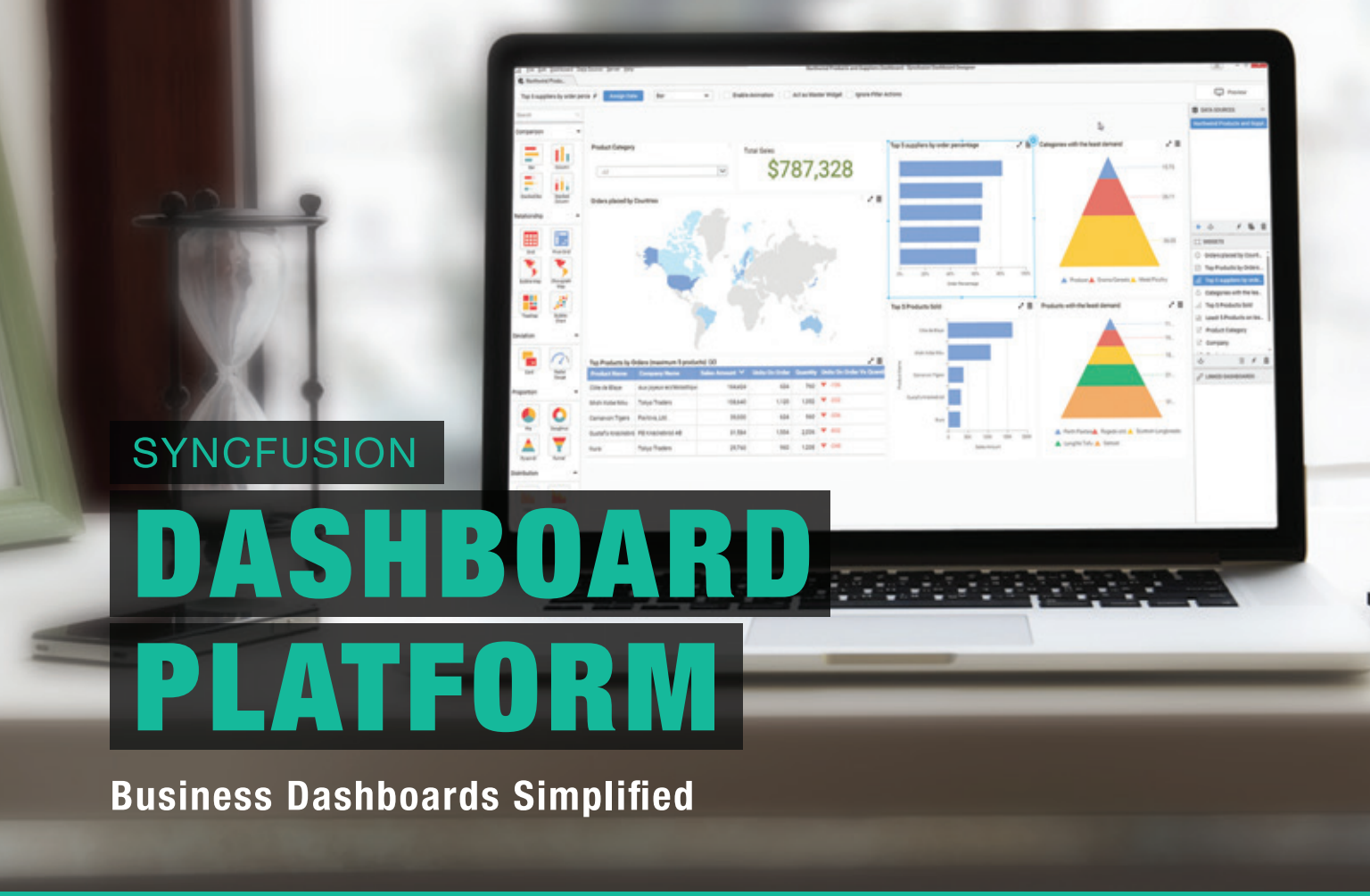
Multi-platform report viewers

Trusted by a worldwide customer base

Easy licensing

Download your free trial at
activereports.grapecity.com

GrapeCity ActiveReports



SYNCFUSION

DASHBOARD PLATFORM

Business Dashboards Simplified

- The complete solution for creating and sharing interactive business dashboards.
- Includes a user-friendly drag-and-drop designer and widgets to visualize your dashboards.
- View, manage, and share dashboards through a web interface.
- Integrates seamlessly with the Syncfusion Big Data Platform.
- Connects to commonly used data sources like Microsoft Excel, CSV, SQL Server, and Spark SQL.

Deploy to 50 users for only **$1,995** per year

FREE COMMUNITY LICENSE AVAILABLE!

Ready to get started?

Download a free trial

www.syncfusion.com/msdndashboard

Syncfusion

ASPOSE

Powerful File APIs that are easy and intuitive to use

Native APIs for
.NET, Java & Cloud

Adding File Conversion and Manipulation to Business Systems

DOC, XLS, JPG, PNG, PDF, BMP, MSG, PPT, VSD, XPS & many other formats.

www.aspose.com

EU Sales: +44 141 416 1112

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

AU Sales: +61 2 8003 5926

Working with Files?

- ✓ CONVERT
- ✓ PRINT
- ✓ CREATE
- ✓ COMBINE
- ✓ MODIFY

100% Standalone - No Office Automation

Scan for
20% Savings!

ASPOSE
Your File Format APIs

ASPOSE.TOTAL

Every Aspose component combined in ONE powerful suite!

Powerful
File Format APIs

- **Aspose.Words**
  DOC, DOCX, RTF, HTML, PDF, XPS & other document formats.
- **Aspose.Cells**
  XLS, XLSX, XLSM, XLTX, CSV, SpreadsheetML & image formats.
- **Aspose.BarCode**
  JPG, PNG, BMP, GIF, TIF, WMF, ICON & other image formats.
- **Aspose.Pdf**
  PDF, XML, XLS-FO, HTML, BMP, JPG, PNG & other image formats.
- **Aspose.Email**
  MSG, EML, PST, EMLX & other formats.
- **Aspose.Slides**
  PPT, PPTX, POT, POTX, XPS, HTML, PNG, PDF & other formats.
- **Aspose.Diagram**
  VSD, VSDX, VSS, VST, VSX & other formats.

... and many others!

Aspose.Total for .NET Aspose.Total for Java Aspose.Total for Cloud

Get your FREE evaluation copy at www.aspose.com

.NET Java Cloud

Aspose.Cells

Work with spreadsheets and data without depending on Microsoft Excel

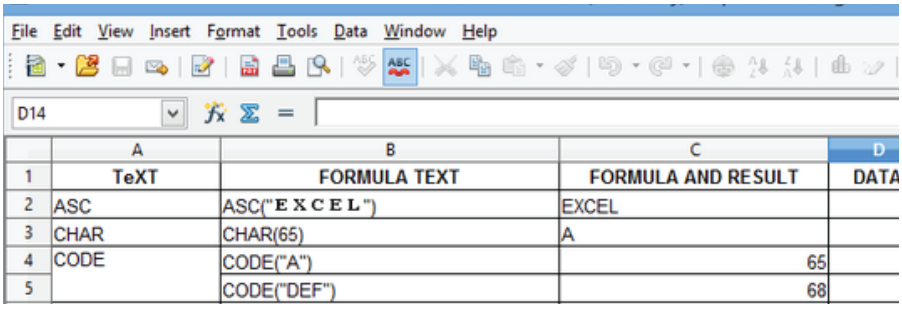
- Solution for spreadsheet creation, manipulation and conversion.
- Import and export data.

ASPOSE.CELLS IS A PROGRAMMING API that allows developers to create, manipulate and convert Microsoft Excel spreadsheet files from within their own applications. Its powerful features make it easy to convert worksheets and charts to graphics or save reports to PDF.

A flexible API for simple and complex spreadsheet programming.

Aspose.Cells speeds up working with Microsoft Excel files. The API is a flexible tool for simple tasks such as file conversion, as well as complex tasks like building models. Developers control page layout, formatting, charts and formulas. They can read and write spreadsheet files and save out to a wide variety of image and text file formats.

Fast and reliable, Aspose.Cells saves time and effort compared to using Microsoft Office Automation.



	A	B	C	D
1	TeXT	FORMULA TEXT	FORMULA AND RESULT	DATA
2	ASC	ASC("EXCEL")	EXCEL	
3	CHAR	CHAR(65)	A	
4	CODE	CODE("A")	65	
5		CODE("DEF")	68	

Aspose.Cells lets developers work with data sources, formatting, even formulas.

Common Uses

- Building dynamic reports on the fly.
- Creating Excel dashboards with charts and pivot tables.
- Rendering and printing spreadsheets and graphics with high fidelity.
- Exporting data to, or importing from, Excel spreadsheets.
- Generating, manipulating and editing spreadsheets.
- Converting spreadsheets to images or other file formats.

Key Features

- A complete spreadsheet manipulation solution.
- Flexible data visualization and reporting.
- Powerful formula engine.
- Complete formatting control.

Supported File Formats

XLS, XLSX, XLSM, XMPS, XLTX, XLTM, ODS, SpreadsheetML, tab delim., CSV, TXT, PDF, HTML, and many image formats including TIFF, JPEG, PNG and GIF.

Format support varies across platforms.

Platforms

Pricing Info					
	Standard	Enhanced		Standard	Enhanced
Developer Small Business	$999	$1498	Site Small Business	$4995	$7490
Developer OEM	$2997	$4494	Site OEM	$13986	$20972

The pricing info above is for .NET: prices for other platforms may differ. For the latest, contact sales.

Aspose.Cells for

.NET, Java, Cloud & more

File Formats

XLS XLSX TXT PDF HTML CSV TIFF PNG JPG BMP SpreadsheetML and many others.

Spreadsheet Manipulation

Aspose.Cells lets you create, import, and export spreadsheets and also allows you to manipulate contents, cell formatting, and file protection.

Creating Charts

Aspose.Cells comes with complete support for charting and supports all standard chart types. Also, you can convert charts to images.

Graphics Capabilities

Easily convert worksheets to images as well as adding images to worksheets at runtime.

No Office Automation

Aspose.Cells does not require Microsoft Office to be installed on the machine in order to work.

Get your FREE Trial at
http://www.aspose.com

ASPOSE

Your File Format APIs

Aspose.Words

Program with word processing documents independently of Microsoft Word

- Solution for document creation, manipulation and conversion.
- Advanced mail merge functionality.

ASPOSE.WORDS IS AN ADVANCED PROGRAMMING API that lets developers perform a wide range of document processing tasks with their own applications. Aspose.Words makes it possible to generate, modify, convert, render and print documents without Microsoft Office Automation. It provides sophisticated and flexible access to, and control over, Microsoft Word files.

Generate, modify, convert, render and print documents without Microsoft Office Automation.

Aspose. Words is powerful, user-friendly and feature rich. It saves developers time and effort compared to using Microsoft Office Automation and makes gives them powerful document management tools.

Aspose.Words makes creating, changing and converting DOC and other word processing file formats fast and easy.

	Table			
	Column 1	**Column 2**	**Column 3**	**Column 4**
Row 1	Cell 1	Cell 2	Cell 3	Cell 4
Row 2	Cell 1	Cell 2	Cell 3	
Row 3	Cell 1	Cell 2		

Aspose.Words has sophisticated controls for formatting and managing tables and other content.

Common Uses

- Generating reports with complex mail merging; mail merging images.
- Populating tables and documents with data from a database.
- Inserting formatted text, paragraphs, tables and images into Microsoft Word documents.
- Adding barcodes to documents.
- Inserting diagrams and watermarks into Word documents.
- Formatting date and numeric fields.

Key Features

- A complete Microsoft Word document manipulation solution.
- Extensive mail merge features.
- Complete formatting control.
- High-fidelity conversion, rendering and printing.

Supported File Formats

DOC, DOCX, ODT, OOXML, XML, HTML, XHTML, MHTML, EPUB, PDF, XPS, RTF, and a number of image formats, including TIFF, JPEG, PNG and GIF.

Format support varies across platforms.

Platforms

Pricing Info					
	Standard	Enhanced		Standard	Enhanced
Developer Small Business	$999	$1498	Site Small Business	$4995	$7490
Developer OEM	$2997	$4494	Site OEM	$13986	$20972

The pricing info above is for .NET: prices for other platforms may differ. For the latest, contact sales.

Case Study: Aspose.Words for .NET

ModulAcht e.K. - using Aspose.Words for .NET to convert from DOCX to PDF.

MODULACHT IS A SOFTWARE DEVELOPMENT TEAM WHICH CREATES INDIVIDUAL SOFTWARE for small businesses. Mostly we develop web applications including web UI and web-services, but we are also familiar with Windows Forms and Windows Services applications based on .NET.

Problem

For our main customer, we are developing the operating system they will use to administer the buying and selling of cars. With a need to generate documents easily, one of the main requirements was to have an easy-to-use template system.

> "The really quick and competent support of Aspose helped us to solve some initial problems."

Looking for a Solution

We searched on the internet for DOCX to PDF converters, which is not as easy as it sounds. After filtering all the Interop wrappers only a handful of components remained to be tested. At the end only Aspose.Words for .NET created a result which really looks like the input DOCX. The really quick and competent support of Aspose helped us to solve some initial problems.

Implementation

Aspose.Words for .NET was the 4th component we tested. On our development machine, everything worked great, but after moving the code on to our test-server-machine, the resulting PDF did not look like the original DOCX file. Adjusting the settings didn't help so we decided give the support team of Aspose a try.

After a short discussion in the live chat we started a new thread including a description, the input and the output file, in the Aspose.Words forum. Within less than 24 hours one of the support-team member told us that we would have to check whether the font we used in the DOCX file was available on the server machine, which it was not. After changing the font, the whole PDF looks exactly the same as the DOCX file.

Outcome

Choosing Aspose.Words for .NET meant an intuitive and easy to use software component and also getting a really friendly and straightforward software partner which is ready to help if you need help.

Next Steps

After getting our Test-Driver ready we will implement the template engine in our customer's software. Aspose.Words for .NET functionality will be used on many different places in this software to convert files into the PDF format.

This is an extract from a case study on our website. For the full version, go to: www.aspose.com/corporate/customers/case-studies.aspx

After converting, our PDF looks exatly the same as the DOCX file.

Open, Create, Convert, Print & Save Files

from within your *own* applications.

ASPOSE.TOTAL

allows you to process these file formats:

- Word documents
- Excel spreadsheets
- PowerPoint presentations
- PDF documents
- Project documents
- Visio documents
- Outlook emails
- OneNote documents

DOC XLS PPT PDF EML
PNG XML RTF HTML VSD
BMP & barcode images.

ASPOSE
Your File Format APIs

Contact Us:
US: +1 888 277 6734
EU: +44 141 416 1112
AU: +61 2 8003 5926
sales@aspose.com

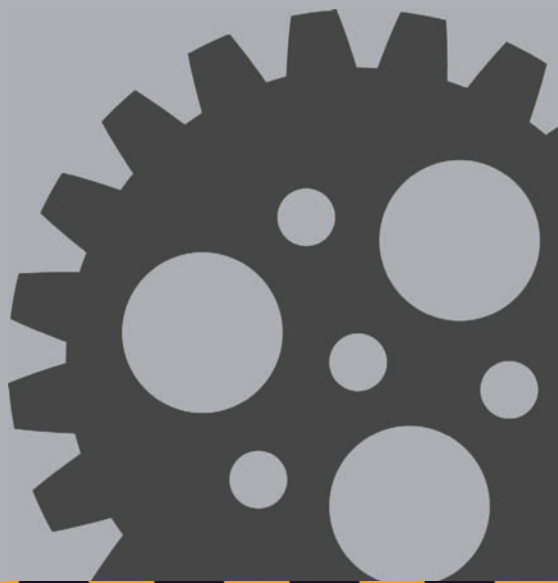
Helped over 11,000 companies and over 250,000 users work with documents in their applications.

.NET, Java, and Cloud

Your File Format APIs

GET STARTED NOW

- Free Trial
- 30 Day Temp License
- Free Support
- Community Forums
- Live Chat
- Blogs
- Examples
- Video Demos



Adding File Conversion and Manipulation to Business Systems

How often do people in your organization complain that they can't get information in the file format and layout they want? Converting documents from one format to another without losing layout and formatting should be simple, but it can be frustrating for both users and developers.

EXTRACTING DATA FROM A DATABASE AND DELIVERING IT TO THE SALES TEAM AS A REPORT, complete with charts and corporate branding, is fine. Until the sales team says that they want it as a Microsoft Excel file, and could you add a dashboard?

Using information from online forms in letters that can are printed and posted is easy. But what if you also want to add tracking barcodes and archive a digital copy as a PDF?

> Aspose creates APIs that work independently of Microsoft Office Automation.

Ensuring that your business system supports all the different Microsoft Office file formats your users want can be difficult. Sometimes the native file format support of your system lets you down. When that is the case, use tools that extend that capability. A good tool can save you time and effort.

Document Conversion Options

Building your own solution: Time-consuming and costly, this option is only sensible if the solution you develop is central to your business.

Using Microsoft Office Automation: Microsoft Office Automation lets you use Microsoft Office programs server-side. It is not how the Office products were designed to be used. It can work well but you might notice issues with the stability, security and speed of the system, as well as cost.

Using an API: The API market has lots of free and commercial solutions, some very focused, some feature-rich. An API integrates with your code and gives you access to a range of new features.

Look to Aspose

Aspose are API experts. We create APIs, components and extensions that work independently of Microsoft Automation to extend a platform's native file format manipulation capabilities.

Aspose have developed APIs for .NET, Java, Cloud and Android that lets developers convert, create and manipulate Microsoft Office files – Microsoft Word, Excel, PowerPoint, Visio and Project – and other popular business formats, from PDFs and images to emails. We also have APIs for working with images, barcodes and OCR. The APIs are optimised for stability, speed and ease of use. Our APIs save users weeks, sometimes months, of effort.

Finding the Right Tool

To find the product that's right for you, take a systematic approach:

- List must-have and nice-to-have features.
- Research the market.
- Ask for recommendations.
- Select a few candidates .
- Run trials.
- Evaluate
  - ease of use,
  - support and documentation,
  - performance, and
  - current and future needs.

Aspose.BarCode

A complete toolkit for barcode generation and recognition

- Generate barcodes with customer defined size and color.
- Recognize a large number of barcode types from images.

ASPOSE.BARCODE IS A ROBUST AND RELIABLE BARCODE GENERATION AND RECOGNITION API that allows developers to add barcode generation and recognition functionality to their applications quickly and easily.

Aspose.BarCode supports most established barcode specifications. It can export generated barcodes to multiple image formats, including BMP, GIF, JPED, PNG and TIFF.

Aspose.BarCode gives you full control over every aspect of the barcode image, from background and bar color, through image quality, rotation angle, X-dimension, captions, and resolution.

Robust and reliable barcode generation and recognition.

Aspose.BarCode can read and recognize most common 1D and 2D barcodes from any image and at any angle. Filters help developers clean up difficult to read images to improve recognition.

Aspose.BarCode offers a large number of symbologies and formatting options.

Common Uses

- Generating and recognizing barcode images.
- Printing barcode labels.
- Enhancing workflow by adding barcode functionality.
- Using recognition functions to drive real-life work processes.

Key Features

- Barcode generation and recognition.
- Comprehensive support for 1D and 2D symbologies.
- Image processing for improved recognition.

Supported File Formats

JPG, TIFF, PNG, BMP, GIF, EMF, WMF, EXIP and ICON.

Format support varies across platforms.

Supported Barcodes

Linear: EAN13, EAN8, UPCA, UPCE, Interleaved2of5, Standard2of5, MSI, Code11, Codabar, EAN14(SCC14), SSCC18, ITF14, Matrix 2 of 5, PZN, Code128, Code39 Extended, Code39 Standard, OPC, Code93 Extended, Code93 Standard, IATA 2 of 5, GS1Code128, ISBN, ISMN, ISSN, ITF6, Pharmacode, DatabarOmniDirectional, VIN, DatabarTruncated, DatabarLimited, DatabarExpanded, PatchCode, Supplement **2D**: PDF417, MacroPDF417, DataMatrix, Aztec, QR, Italian Post 25, Code16K, GS1DataMatrix **Postal**: Postnet, Planet, USPS OneCode, Australia Post, Deutsche Post Identcode, AustralianPosteParcel, Deutsche Post Leticode, RM4SCC, SingaporePost, SwissPostParcel

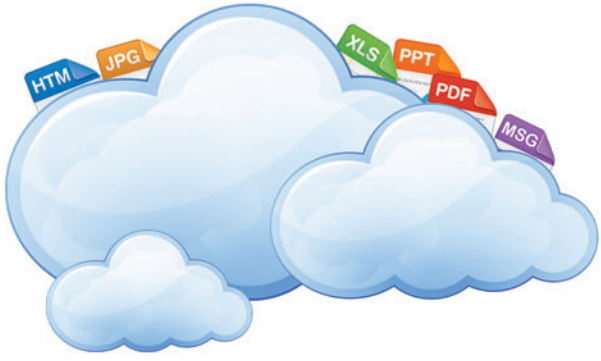
Platforms

Pricing Info					
	Standard	Enhanced		Standard	Enhanced
Developer Small Business	$599	$1098	Site Small Business	$2995	$5490
Developer OEM	$1797	$3294	Site OEM	$8386	$15372

The pricing info above is for .NET: prices for other platforms may differ. For the latest, contact sales.

Aspose *for* Cloud

The easiest API to Create, Convert & Automate Documents in the cloud.



Convert
Create
Render
Combine
Modify

without installing anything!

Aspose.Words for Cloud Create and convert docs Manipulate text Render documents Annotate	**Aspose.Cells for Cloud** Create spreadsheets Convert spreadsheets Manipulate cells and formulas Render spreadsheets
Aspose.Slides for Cloud Create presentations Manage slides Edit text and images Read and convert	**Aspose.Pdf for Cloud** Create and convert PDFs Manipulate text, images Add pages, split, encrypt Manage stamps
Aspose.Email for Cloud Create, update, and convert messages Extract attachments Use with any language	**Aspose.BarCode for Cloud** Generate barcodes Read barcodes Set attributes Multiple image formats

Free Evaluation at www.aspose.com

- +44 141 416 1112
- +1 888 277 6734
  sales@aspose.com
- +61 2 8003 5926

Aspose.Email

Work with emails and calendars without Microsoft Outlook

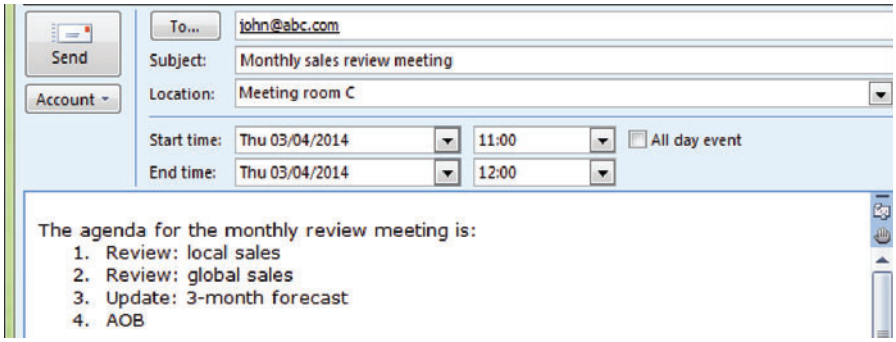
- Complete email processing solution.
- Message file format support.

ASPOSE.EMAIL IS AN EMAIL PROGRAMMING API that allows developers to access and work with PST, EML, MSG and MHT files. It also offers an advanced API for interacting with enterprise mail systems like Exchange and Gmail.

Aspose.Email can work with HTML and plain text emails, attachments and embedded OLE objects. It allows developers to work against SMTP, POP, FTP and Microsoft Exchange servers. It supports mail merge and iCalendar features, customized header and body, searching archives and has many other useful features.

Aspose.Email works with HTML and plain text emails, attachments and embedded OLE objects.

Aspose.Email allows developers to focus on managing email without getting into the core of email and network programming. It gives you the controls you need.



Aspose.Email lets your applications work with emails, attachments, notes and calendars.

Common Uses

- Sending email with HTML formatting and attachments.
- Mail merging and sending mass mail.
- Connecting to POP3 and IMAP mail servers to list and download messages.
- Connecting to Microsoft Exchange Servers to list, download and send messages.
- Create and update tasks using iCalendar.
- Load from and save messages to file or stream (EML, MSG or MHT formats).

Key Features

- A complete email processing solution.
- Support for MSG and PST formats.
- Microsoft Exchange Server support.
- Complete recurrence pattern solution.

Supported File Formats

MSG, MHT, OST, PST, EMLX, TNEF, and EML.

Format support varies across platforms.

Platforms

Pricing Info					
	Standard	Enhanced		Standard	Enhanced
Developer Small Business	$599	$1059	Site Small Business	$2995	$5490
Developer OEM	$1797	$3294	Site OEM	$8386	$15372

The pricing info above is for .NET: prices for other platforms may differ. For the latest, contact sales.

Aspose.Pdf

Create PDF documents without using Adobe Acrobat

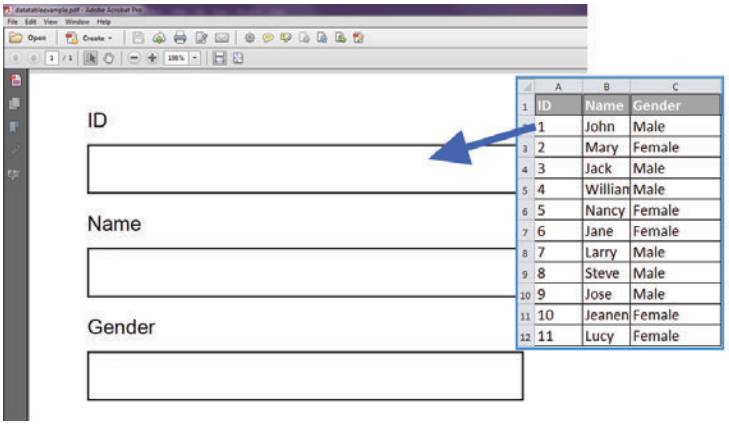
- A complete solution for programming with PDF files.
- Work with PDF forms and form fields.

ASPOSE.PDF IS A PDF DOCUMENT CREATION AND MANIPULATION API that developers use to read, write and manipulate PDF documents without using Adobe Acrobat. Aspose.Pdf is a sophisticated product that integrates with your application to add PDF capabilities.

Aspose.Pdf offers a wealth of features that lets developers compress files, create tables, work with links, add and remove security, handle custom fonts, integrate with external data sources, manage bookmarks, create table of contents, create forms and manage form fields.

> Read, write and manipulate PDF documents independently of Adobe Acrobat.

It helps developers add, work with attachments, annotations and PDF form data, add, replace or remove text and images, split, concatenate, extract or inset pages, and print PDF documents.



Aspose.Pdf can be used to automatically complete PDF forms with external data.

Common Uses

- Creating and editing PDF files.
- Inserting, extracting, appending, concatenating and splitting PDFs.
- Working with text, images, tables, images, headers, and footers.
- Applying security, passwords and signatures.
- Working with forms and form fields.

Key Features

- PDF creation from XML or XLS-FO documents.
- PDF form and field support.
- Advanced security and encryption.
- High-fidelity printing and conversion.
- Supported File Formats
- PDF, PDF/A, PDF/A_1b, PCL, XLS-FO, LaTeX, HTML, XPS, TXT and a range of image formats.

Format support varies across platforms.

Platforms

Pricing Info					
	Standard	Enhanced		Standard	Enhanced
Developer Small Business	$799	$1298	Site Small Business	$3995	$6490
Developer OEM	$2397	$3894	Site OEM	$11186	$18172

The pricing info above is for .NET: prices for other platforms may differ. For the latest, contact sales.

Aspose.Pdf

.Net, Java & Cloud

File Formats

PDF XPS ePUB HTML XML XLS TXT DOC XSL-FO & other image file formats.

Create and Manipulate PDFs

Create new or edit/manipualte existing PDFs.

Form Field Features

Add form fields to your PDFs. Import and export form fields data from select file formats.

Table Features

Add tables to your PDFs with formatting such as table border style, margin and padding info, column width and spanning options, and more.

Get started today at www.aspose.com

Conversion is Fast And High-Fidelity

Aspose.Note for .NET

Aspose.Note for .NET is an API that lets developers convert Microsoft OneNote pages to a variety of file formats, and extract the text and document information.

Conversion is fast and high-fidelity. The output looks like the OneNote page, no matter how complex the formatting or layout.

Aspose.Note works independently of Office Automation and does not require Microsoft Office or OneNote to be installed.

Product	Benefit	Supported Platforms
Aspose.Note for .NET	Modify, convert, render and extract text and images from Microsoft OneNote files without relying on OneNote or other libraries.	.NET Framework 2.0, 3.0, 3.5, 4.0, 4.0 CP

Features

File Formats and Conversion		Rendering and Printing	Document Management
Microsoft OneNote 2010, 2010 SP1, 2013	Load, Save	Save as Image (BMP, GIF, JPG, PNG)	• Extract text • Get the number of pages in a document. • Get page information. • Extract images. • Get image information from a document. • Replace text in document.
PDF	Save	Save as PDF	
Images (BMP, GIF, JPG, PNG)	Save		

Aspose.Imaging

Create Images from scratch.

- Load existing images for editing purposes.
- Render to multiple file formats.

ASPOSE.IMAGING IS A CLASS LIBRARY that facilitates the developer to create Image files from scratch or load existing ones for editing purpose. Also, Aspose.Imaging provides the means to save the created or edited Image to a variety of formats. All of the above mentioned can be achieved without the need of an Image Editor. It works independent of other applications and although Aspose.Imaging allows you to save to Adobe PhotoShop® format (PSD), you do not need PhotoShop installed on the machine.

Aspose.Imaging is flexible, stable and powerful. It's many features and image processing routines should meet most imaging requirements. Like all Aspose file format components, Aspose. Imaging introduces support for an advanced set of drawing features along with the core functionality. Developers can draw on Image surface either by manipulating the bitmap information or by using the advanced functionality like Graphics and Paths.

Aspose.Imaging allows creation and manipulation of images.

Create images from scratch. or load existing ones...

Common Uses

- Create images from scratch.
- Load and Edit existing images.
- Export images to a variety of formats.
- Adding watermark to images.
- Export CAD drawings to PDF & raster image formats.
- Crop, resize & RotateFlip images.
- Extract frames from multipage TIFF image.

Key Features

- Create, edit, and save images
- Multiple file formats
- Drawing features
- Export images

Supported File Formats

BMP, JPG, TIFF, GIF, PNG, PSD, DXF, DWG, and PDF.

Platforms

Pricing Info					
	Standard	Enhanced		Standard	Enhanced
Developer Small Business	$399	$898	Site Small Business	$1995	$4490
Developer OEM	$1197	$2694	Site OEM	$5586	$12572

The pricing info above is for .NET.

Aspose.Slides

Work with presentations without using Microsoft PowerPoint

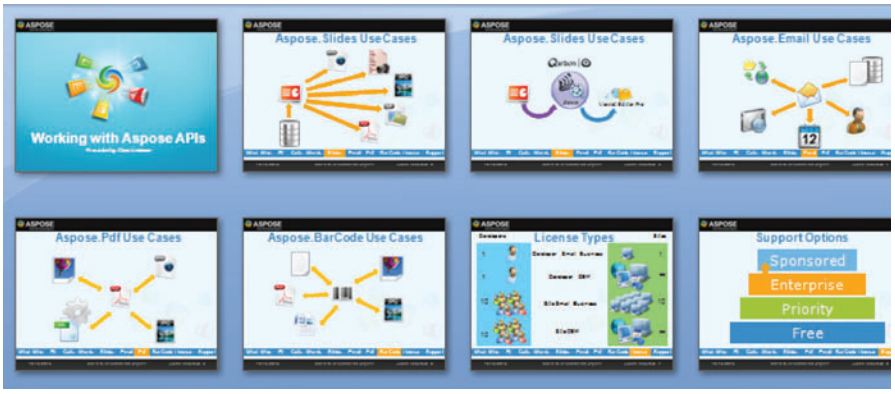
- Complete solution for working with presentation files.
- Export presentations and slides to portable or image formats.

ASPOSE.SLIDES IS A FLEXIBLE PRESENTATION MANAGEMENT API that helps developers read, write and manipulate Microsoft PowerPoint documents. Slides and presentations can be saved to PDF, HTML and image file formats without Microsoft Office Automation.

Aspose.Slides offers a number of advanced features that make it easy to perform tasks such as rendering slides, exporting presentations, exporting slides to SVG and printing. Developers use Aspose.Slides to build customizable slide decks, add or remove standard graphics and automatically publish presentations to other formats.

Aspose.Slides gives you the tools you need to work with presentation files.

Aspose.Slides gives developers the tools they need to work with presentation files. It integrates quickly and saves time and money.



Aspose.Slides has advanced features for working with every aspect of a presentation.

Common Uses

- Creating new slides and cloning existing slides from templates.
- Handling text and shape formatting.
- Applying and removing protection.
- Exporting presentations to images and PDF.
- Embedding Excel charts as OLE objects.
- Generate presentations from database.

Key Features

- A complete presentation development solution.
- Control over text, formatting and slide elements.
- OLE integration for embedding external content.
- Wide support for input and output file formats.

Supported File Formats

PPT, POT, PPS, PPTX, POTX, PPSX, ODP, PresentationML, XPS, PDF and image formats including TIFF and JPG.

Format support varies across platforms.

Platforms

Pricing Info					
	Standard	Enhanced		Standard	Enhanced
Developer Small Business	$799	$1298	Site Small Business	$3995	$6490
Developer OEM	$2397	$3894	Site OEM	$11186	$18172

The pricing info above is for .NET: prices for other platforms may differ. For the latest, contact sales.

Support Services

Get the assistance you need, when you need it, from the people who know our products best.

- Use experienced Aspose developers for your projects
- Get the level of support that suits you and your team

NO ONE KNOWS OUR PRODUCTS AS WELL AS WE DO.

We develop them, support them and use them. Our experience is available to you, whether you want us to develop a solution for you, or you just need a little help to solve a particular problem.

Consulting

Aspose's developers are expert users of Aspose APIs. They understand how to use our products and have hands-on experience of using them for software development. Aspose's developers are skilled not just with Aspose tools but in a wide range of programming languages, tools and techniques.

Aspose's file format experts are here to help you with a project or your support questions

When you need help to get a project off the ground, Aspose's developers can help.

Work with the most experienced Aspose developers in the world.

Consulting Benefits

- Use Aspose engineers to work on your products
- Get peace of mind from a fully managed development process
- Get a custom-built solution that meets your exact needs

Support Options

Free

Everyone who uses Aspose products have access to our free support. Our software developers are on stand-by to help you succeed with your project, from the evaluation to roll-out of your solution.

Priority

If you want to know when you'll hear back from us on an issue, and know that your issue is prioritized, Priority Support is for you. It provides a more formal support structure and has its own forum that is monitored by our software engineers.

Enterprise

Enterprise customers often have very specific needs. Our Enterprise Support option gives them access to the product development team and influence over the roadmap. Enterprise Support customers have their own, dedicated issue tracking system.

Pricing Info
Each consulting project is evaluated individually; no two projects have exactly the same requirements.
To see the Priority and Enterprise support rates, refer to the product price list, or contact our sales team.

We're Here to Help You

Aspose has 4 Support Services to best suit your needs

Free Support	Support Forums with no Charge
Priority Support	24 hour response time in the week, issue escalation, dedicated forum
Enterprise Support	Communicate with product managers, influence the roadmap
Sponsored Support	Get the feature you need built now

Technical Support is an issue that Aspose takes very seriously. Software must work quickly and dependably. When problems arise, developers need answers in a hurry. We ensure that our clients receive useful answers and solutions quickly.

Email • Live Chat • Forums

Contact Us

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

EU Sales: +44 141 416 1112

AU Sales: +61 2 8003 5926