THE MICROSOFT JOURNAL FOR DEVELOPERS

MARCH 2017 VOL 32 NO 3

msdn magazine

DevExpress

Your Next Great Desktop App Starts Here

From apps that replicate the look and feel of Microsoft Office® to high-powered data mining and decision support systems for your enterprise, DevExpress desktop controls will help you build your best, without limits or compromise.

Download your free 30-day trial
an experience the DevExpress difference today.

devexpress.com/try

THE MICROSOFT JOURNAL FOR DEVELOPERS

MARCH 2017 VOL 32 NO 3

msdn magazine

Verify Source Code Files in Visual Studio.....12

COLUMNS

Microsoft

WE'RE CHANGING THE WAY YOU LOOK AT REPORTING

The Text Control Reporting Framework combines powerful reporting features with an easy-to-use, MS Word compatible word processor.

Our award-winning developer libraries are completely independent from MS Word or any other third-party application and can be completely integrated into your business application.

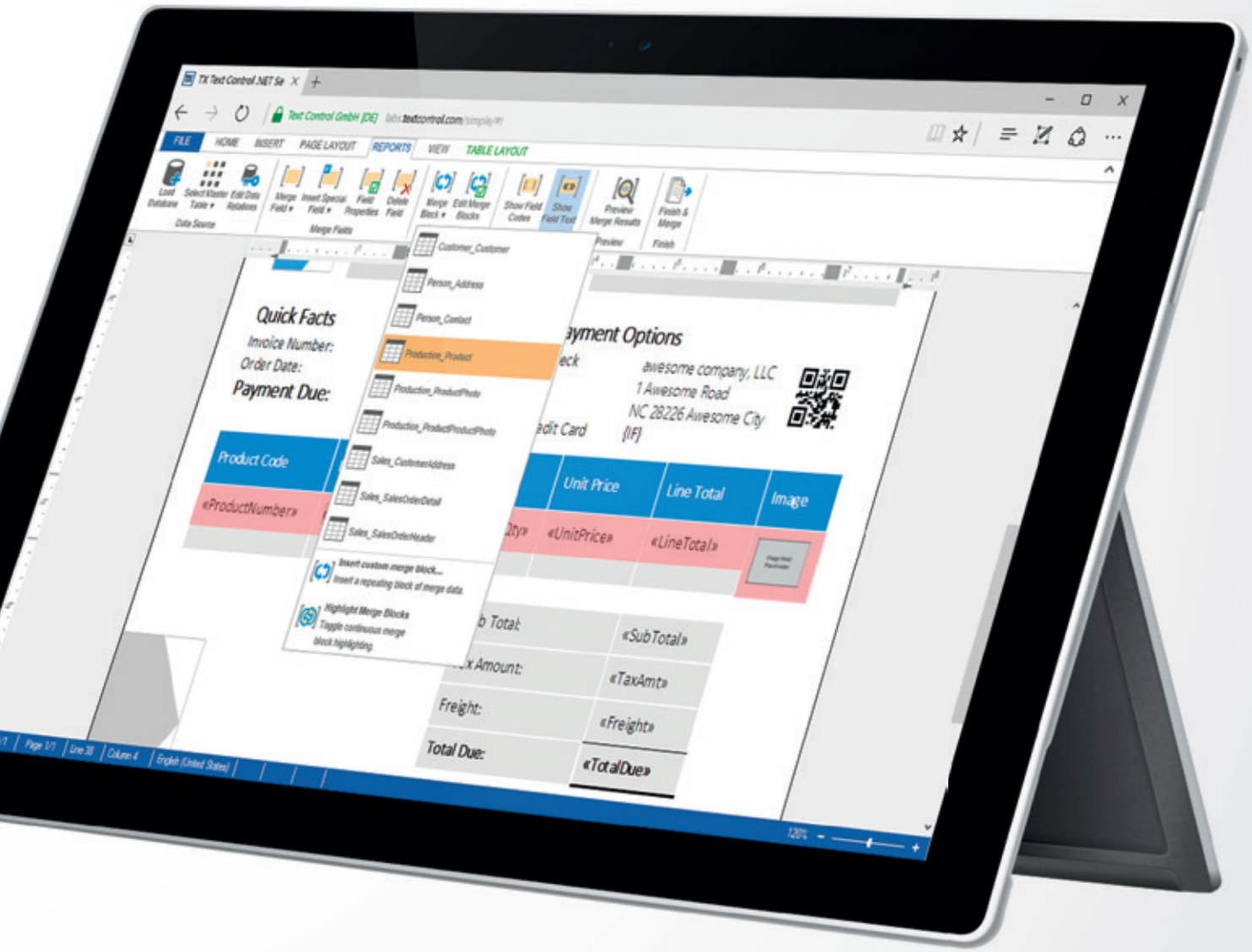
Follow the trend and switch to flow type layout reporting.

www.textcontrol.com www.reporting.cloud

Microsoft .NET WINDOWS FORMS | WPF | ASP.NET MVC | ASP.NET AJAX | CLOUD WEB API

Visual Studio Partner

REPORTING LIBRARIES FOR WINDOWS, WEB, MOBILE AND CLOUD APPLICATIONS



TEXT CONTROL

msdn magazine

MARCH 2017 VOLUME 32 NUMBER 3

ID STATEMENT MSDN Magazine (ISSN 1528-4859) is published 13 times a year, monthly with a special issue in November by 1105 Media, Inc., 9201 Oakdale Avenue, Ste. 101, Chatsworth, CA 91311. Periodicals postage paid at Chatsworth, CA 91311-9998, and at additional mailing offices. Annual subscription rates payable in US funds are: U.S. $35.00, International $60.00. Annual digital subscription rates payable in U.S. funds are: U.S. $25.00, International $25.00. Single copies/back issues: U.S. $10, all others $12. Send orders with payment to: MSDN Magazine, P.O. Box 3167, Carol Stream, IL 60132, email MSDNmag@1105service.com or call (847) 763-9560. POSTMASTER: Send address changes to MSDN Magazine, P.O. Box 2166, Skokie, IL 60076. Canada Publications Mail Agreement No: 40612608. Return Undeliverable Canadian Addresses to Circulation Dept. or XPO Returns: P.O. Box 201, Richmond Hill, ON L4B 4R5, Canada.

CORPORATE ADDRESS 1105 Media, 9201 Oakdale Ave. Ste 101, Chatsworth, CA 91311 www.1105media.com

MEDIA KITS Direct your Media Kit requests to Chief Revenue Officer Dan LaBianca, 972-687-6702 (phone), 972-687-6799 (fax), dlabianca@1105media.com

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

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

Reaching the Staff

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

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

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

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

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

Microsoft

BPA WORLDWIDE

1906 2006 100 AMERICAN BUSINESS MEDIA

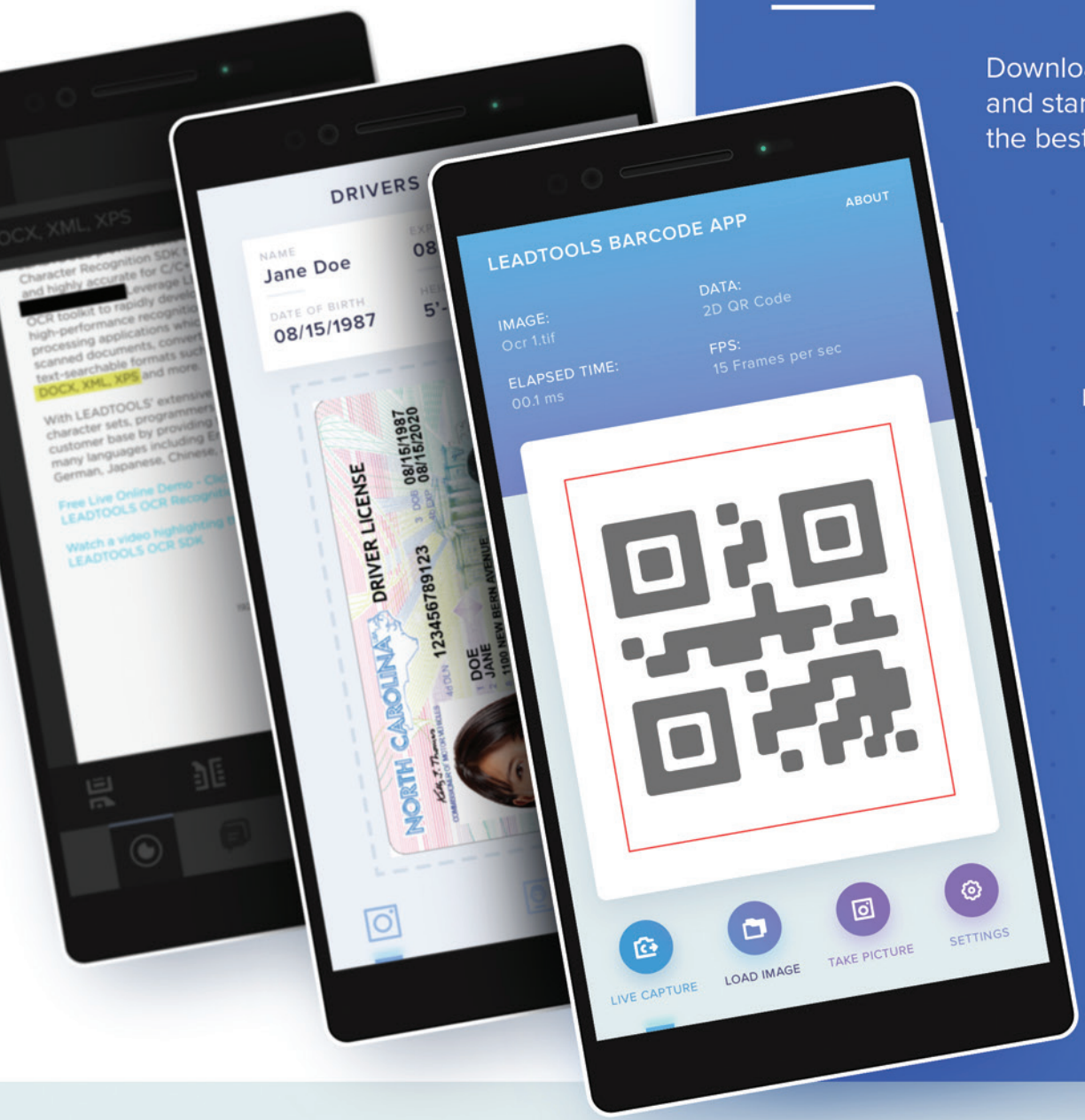
WPA Western Publications Association

LEADTOOLS®

THE WORLD LEADER IN IMAGING SDKs

WORLD-CLASS MOBILE RECOGNITION SDK

Quickly and accurately extract data from any document or image with LEADTOOLS. Developers using the SDK easily add advanced OCR, Barcode, Forms, Driver's License, Passport, Check, and Credit Card recognition functionality into their applications.



Download the SDK today and start developing with the best in:

- RELIABILITY
- ACCURACY
- SPEED

Fully-functional sample apps built with LEADTOOLS are also available for download from **Google Play**, **App Store**, and **Microsoft Store**.

DOWNLOAD OUR 60 DAY EVALUATION
WWW.LEADTOOLS.COM

LEAD TECHNOLOGIES INCORPORATED

SALES@LEADTOOLS.COM
800.637.1840

MICHAEL DESMOND

Betting on Bots

When Microsoft Chief Executive Officer Satya Nadella introduced the concept of "conversations as a platform" at the Build 2016 conference, he posited a future era of intelligent bots that would transform the way things get done. Bots would surface in ubiquitous channels like chat and communication clients to provide seamless access to services and business logic that until now had to be deployed as mobile apps or accessed via the Web. Just as important, the use of natural language UIs and intelligent backend services would combine to enable human-centric interaction via voice, text and screen input.

> There's a lot to be excited about, but coders entering the world of bot development face some adjustments.

These are topics we've tackled before. In December, Srikantan Sankaran explored bot development in his feature, "Use Bot Framework for Anytime, Anywhere Access to Application Data" (msdn.com/magazine/mt790202), and Ashish Sahu dove into the Microsoft Language Understanding Intelligence Service (LUIS) in his January issue feature titled, "Enable Natural Language Interaction with LUIS" (msdn.com/magazine/mt745095). This month Kevin Ashley offers guidance on how to leverage Microsoft platforms like Cognitive Services to enable rich interaction in his article, "Making Bots More Intelligent."

There's a lot to be excited about, but coders entering the world of bot development face a few adjustments. Ashley cautions developers not to try to do too much in a bot project. The advantages of speech- and text-driven conversational UI can break down, he says, when software must contend with open-ended dialog and a vast array of possible outcomes. He also urges developers to take full advantage of the unparalleled reach of bot software by leveraging as many channels as possible across mobile, fixed and screenless devices.

Those adjustments extend to the architecture model of bot software, says Maarten Van De Bospoort, who, as principal software development engineer at Microsoft, works with developers and architects at large consumer software companies on technologies like bots, Cognitive Services and Universal Windows Platform apps.

"Since bots are basically Web services, on a tactical level an app developer needs to make a shift from client side to server side," Van De Bospoort says. "Although not as dramatic as 10 years ago, there still are different challenges between the two disciplines, such as threading issues with controls and challenges with state and scalability. A Web developer might have a leg up in that case."

Bots also open a new front in UI development, where the focus shifts from app design and layout to intuitive flow of conversation. Van De Bospoort says developers need to anticipate what users will ask next and avoid deep hierarchies in which users can get lost.

"At the end of the day, a bot needs to help a user accomplish something—whether that's by pressing buttons, talking, sending a picture, or typing full sentences," he says, noting that it's quicker to press buttons than to type the specifics of an order. "If the bot understands the user, it might be a cool experience the first few times, but then it gets cumbersome quickly."

What's your take on Microsoft's vision of a bot-driven future? Will your dev shop be looking to Microsoft Bot Framework as a way to deliver services and enable interaction?

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

Switch to Amyuni PDF

AMYUNI

Create 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

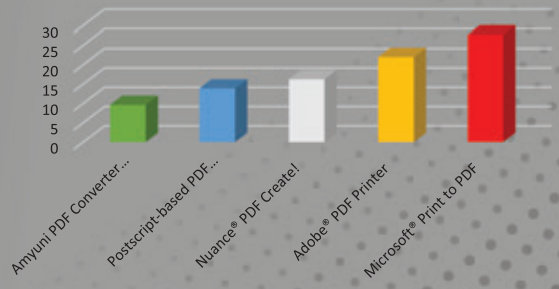
Other Developer Components from Amyuni®

- 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

Dodging Disaster: Job-Networking Gaffes to Avoid

Early in my career, when I was a software engineer at Microsoft, I represented the company at recruiting events, hiring mixers and college career fairs. Too often, I saw potential hires trip themselves up with avoidable mistakes. Here's a list of things to avoid at your next recruiting event:

Short Selling

Be selective in your story telling. If you worked on an otherwise-attractive project that got canceled, there's no need to mention the cancelation part. Also, don't downplay your own past responsibilities, roles or projects by—for example—mentioning that you were the most junior member of the team.

Quality Enthusiasm

When engineers approach me at hiring events, I notice that about 80 percent of them show *no targeted enthusiasm*. That is, they tend to be passive during the conversation and ask rudimentary questions about the job or company. About 15 percent show fake or generic enthusiasm, coming off as insincere or lacking knowledge specific to the company. Only 5 percent of developers show what I'd call real enthusiasm, marked by the ability to directly articulate their goals.

For example, I may identify one attribute that gets me excited about a potential employer—even the most terrible employer has one attractive characteristic. I then articulate this to the recruiter as the reason for my interest. Often, in trying to *express* my enthusiasm, I will automatically *become* enthusiastic.

Poor Story Telling

Don't talk about what your team did in your past projects. Instead, mention what *you* did, even if you did it as part of a team. Everybody knows that you don't work in isolation. The recruiter isn't hiring your team; she's hiring just you.

Developers also mistakenly use phrases like, "I don't remember the details," or, "It was so long ago," when recounting experiences. This is conversational suicide. The recruiter will ignore whatever you say next. Do your best to remember what happened and state it without voicing doubt or hesitation. Most recruiters understand that you don't have a perfect memory. They're more interested in the arc of the story than the exact recounting of every detail.

Hard Sell

Don't *tell* employers why you're a good fit, *show* them. Direct selling falls flat for engineering roles. So, don't say: "I used C# in this previous role. I think it will be really helpful for a role at your company." Instead, say: "Most of my experience so far has been in C#. My most recent project was a large-scale, enterprise-wide deployment of a payroll system that used a C# back end and a SQL Server database." If I've done my research well, and the company actually prizes C# and SQL skills, these words will work their magic.

Bad Timing

Wait to get an offer before you make requests of an employer. Some developers instruct recruiters to place them on a specific product, or worse, tell them the teams they *don't* want to work with. Others want to discuss things like benefits, perks and commute times. A reasonable request or two, stated with politeness and flexibility, is fine, but an interview or hiring event is no place to make job demands or convey your commitment to work-life balance.

Revealed Weakness

Sometimes developers draw undue attention to their own perceived weaknesses, and thus destroy their own candidacies. They may say, "I dropped out of college to take care of my family," or, "While I don't have a CS degree, I've worked as a software engineer for 10 years." Nobody even asked them!

Most of the time, interviewers won't even notice the education section until these candidates mention it. If you're genuinely interested in a job, *assume* that you'll succeed, instead of identifying why you wouldn't. Drawing attention to a weakness magnifies it. Talk only about strengths, and the weaknesses will die of starvation.

Unknown Commodity

Your job as a candidate is to present common ground with the company to which you're talking. If your experience is all in hardware engineering, talk specifically about the software components of your work, emphasize your computer science coursework, and tweak your job descriptions and project selection. Developers sometimes make the mistake of playing up irrelevant or ancient experience, or worthlessly impressive credentials. The recruiter may not care.

Negative Vibes

If what you're about to say about a past colleague or employer could be even *remotely* considered badmouthing, don't say it! It repels recruiters, because if you badmouth somebody else, they know it's only a matter of time until you badmouth them. If you feel compelled to mention something negative, rechannel that restlessness. Mention something positive about the *new* employer instead. ■

KRISHNAN RANGACHARI *is a career coach for software engineers and the founder of ByteshiftResumes.com. Visit RadicalShifts.com for free career success classes for developers.*

DevExpress

DevExpress Spreadsheet for WPF & WinForms

with Chart, Pivot Table and Cell Editor support

Your users already know Microsoft® Excel® and with the new capabilities we've introduced in our Spreadsheet Control, you can create solutions that utilize a "spreadsheet-first" UX for a broad range of use-case scenarios. Imagine sales entry forms that are an actual invoice or an inventory management document with built-in ordering and data entry options.

Give our Spreadsheet a try today and see how easy it is to build Windows® apps that amaze.

Free 30-day trial

devexpress.com/spreadsheet

#UseTheBest

DINO ESPOSITO

Soft Updates with Temporal Tables

The vast majority of today's developers use a classic relational database to store data. It's an approach that worked for decades and still works today, even though alternate and schemaless data stores (collectively known as NoSQL stores) are proving themselves quite effective in a variety of business scenarios. Every time you update an existing table record, you automatically lose track of the state it had before. Overriding existing data hasn't been a big problem for companies so far, but things are changing fast. Today, data is the most valuable asset of every company and the input of business intelligence processes.

In the May and June 2016 installments of this column (msdn.com/magazine/mt703431 and msdn.com/magazine/mt707524, respectively), I discussed a general-purpose approach to take a create, read, update, delete (CRUD) system to the next level through soft updates and deletes. A soft update is a standard update operation, except that the old state of the record is preserved in some way. In doing so, you have an additional API to retrieve the history of each entity created during the system's lifetime.

There aren't many ways in which you can approach the problem of updating and deleting records while keeping track of them. For deletions, that means adding an extra column—typically a Boolean column—to mark the record as deleted. For updates, the most functional approach is to create and maintain a separate history table for each that's relevant to track. Keeping data and history tables in sync requires additional business and data access logic, and a dedicated API to query the history.

Managing historical data in relational tables has been formalized in the ANSI SQL 2011 standard. In its newest version, SQL Server supports a feature called temporal tables that lets you create and manage a shadow historical table for each data table of your choice. This month, I'll delve into SQL Server 2016 temporal tables and their use from within Entity Framework (EF).

Structure of Temporal Tables

Conceptually, the main difference between temporal databases and databases is that classic databases only store data that's true at the current time. Temporal databases, instead, maintain multiple copies of each piece of data. A temporal table appends a few extra time columns to denote when the record reached the given state. **Figure 1** shows how a temporal table looks like in SQL Server 2016 Management Studio.

There are a couple of noticeable things in the figure. One is the child history table, named dbo.BookingsHistory. This table is automatically created by SQL Server every time it processes the T-SQL instruction that would create a temporal table. Developers have only read access to the history table. The other thing to notice in **Figure 1** is the lack of the Delete command in the context menu of the selected table. Once a temporal table is created, any further manipulation on it, whether it happens through the interface of SQL Server Management Studio or programmatically, is strictly controlled and in some cases limited. As examples of this, you can't drop or replicate a temporal table, and limitations also exist on cascading updates and deletions. More information on limitations that affect temporal tables in SQL Server 2016 can be found at bit.ly/2iahP1n.

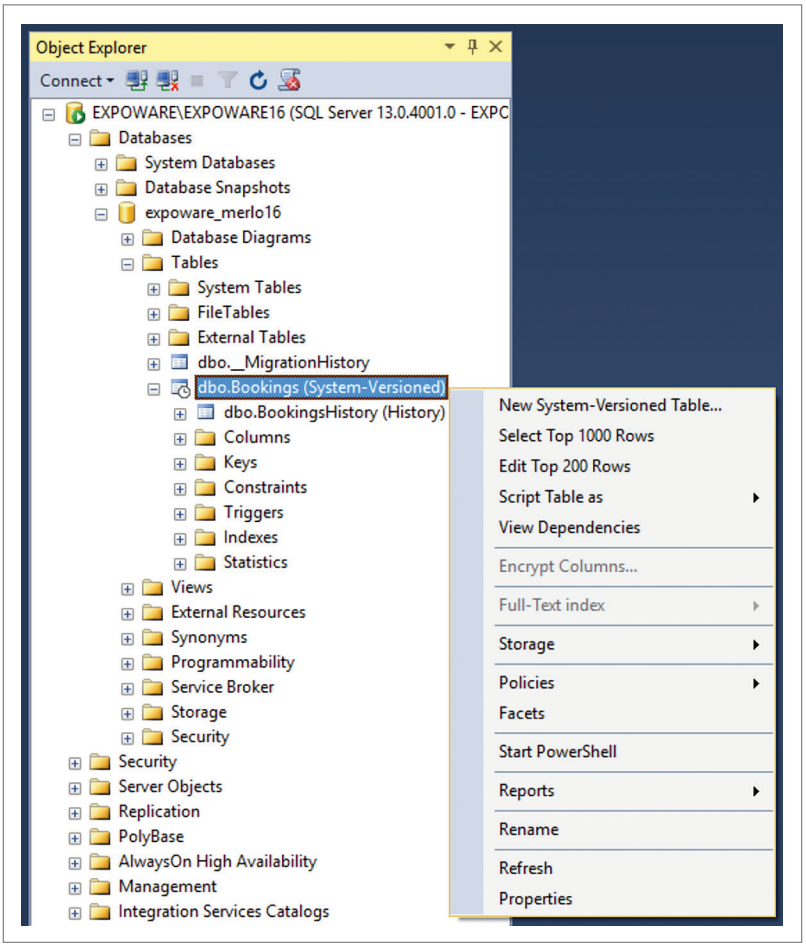


Figure 1 **A Temporal Table in SQL Server 2016**

In SQL Server 2016, you create a temporal table using a special clause at the end of the CREATE TABLE instruction. The status of temporal table boils down to turning on and off the value of the new SYSTEM_VERSIONING setting. In other words, any table can programmatically be turned into a temporal table and then moved back to the original non-temporal status at any time. All aforementioned limitations that affect temporal tables cease to exist the moment in which the SYSTEM_VERSIONING setting is turned off.

Temporal Table and Entity Framework

Many developers today use SQL Server through the services of EF and EF Code First, in particular. At the moment, though, EF provides no special support for temporal tables. Ad hoc support will come in the near future. The good news is that some basic level of support for temporal tables can still be achieved with the current versions of EF 6.x and even more with EF Core. The bad news is that full integration with LINQ-to-Entities is realistically possible to achieve only with low-level changes to the framework, specifically in the way in which the LINQ-to-Entities provider generates the SQL code for the query. If you're a SQL developer instead, then the newest T-SQL language provides all the syntax tools you need to operate temporal tables.

The state of the art of temporal tables for EF developers can be summarized as follows: First, it's pretty easy to create temporal tables in Code First. Second, updates and deletions can be operated via EF in the usual way. Third, queries require ad hoc facilities to be created.

The approach that I've found the most effective to query temporal tables passes through a small set of ad hoc repository methods based on ADO.NET code. It might sound surprising at first, but at the end of the day, if you need temporal tables, then chances are you mostly need to get the history of one particular entity. Take, for example, all the changes of an order or an invoice.

Therefore, in the end, all you need to have is a dedicated and possibly handy FirstOrDefault-like method directly exposed for an aggregate. And a repository class looks to me like a good place to have it.

Figure 2 **Creating a Temporal Table Via Code First**

```
protected override void Seed(DbContext context)
{
  // Grab the SQL Server version number
  var data = context
    .Database
    .SqlQuery<string>(@"select
  left(cast(serverproperty('productversion')
       as varchar), 4)")
    .FirstOrDefault();
  if (data != null)
  {
    var version = data.ToInt();+
    if (version < 13)
      throw new Exception("Invalid version of SQL Server");
  }

  // Prepare the SQL to turn system versioning on SQL Server 2016
  var cmd = String.Format(SqlSystemVersionedDbFormat, "Bookings");
  context.Database.ExecuteSqlCommand(cmd);
}
```

Figure 3 **Creating a Temporal Table**

```
private const string SqlSystemVersionedDbFormat =
  @"ALTER TABLE dbo.{0}
    ADD SysStartTime datetime2(0)
    GENERATED ALWAYS AS ROW START HIDDEN CONSTRAINT
      DF_{0}_SysStart DEFAULT SYSUTCDATETIME(),
    SysEndTime datetime2(0)
    GENERATED ALWAYS AS ROW END HIDDEN CONSTRAINT
      DF_{0}_SysEnd DEFAULT CONVERT(datetime2 (0),
      '9999-12-31 23:59:59'),
    PERIOD FOR SYSTEM_TIME (SysStartTime, SysEndTime)
  ALTER TABLE dbo.{0}
    SET (SYSTEM_VERSIONING = ON (HISTORY_TABLE = dbo.{0}History))";
```

A Temporal-Enabled Initializer

In EF Code First, a new database is created whenever the database doesn't exist and the DbContext class inherits from CreateDatabaseIfNotExists. In doing so, a new table is created for each declared DbSet property. Can you create a temporal table, as well? Today, without ad hoc attributes and syntax features, creating a temporal table is a two-step operation. First, the regular table must be created; this is just the kind of work that Code First usually does. The second step revolves around turning on the SYSTEM_VERSIONING setting. This requires an ad hoc ALTER TABLE statement. **Figure 2** shows a possible implementation of the Seed method of the initializer class that first checks the version of the underlying SQL Server and then alters the state of a previously created Booking table.

The actual T-SQL command you need to create a temporal table (or a system-versioned table, as it's referenced in the jargon of SQL Server 2016), is shown in **Figure 3**.

The {0} placeholders you see in the string in **Figure 3** refer to the name of the actual table. It's Bookings, as shown in **Figure 1**.

The resulting history table is a copy of the main table, plus a couple of datetime2 columns named SysStartTime and SysEndTime. Altogether, the two columns indicate the validity period for that particular state of the record. SysStartTime indicates when the record got a given state and SysEndTime indicates when the validity of that state ceased. Updates and deletes are the database operations that cause the values in SysStartTime and SysEndTime to change.

Updates and Deletes

The logic that keeps that primary table and its history synchronized is backed into the SQL Server 2016 database engine. A new history record is created whenever a record in the primary table is updated, regardless of how you perform the update. In other words, whether you edit the values of a temporal record directly in Management Studio, through stored procedures, ADO.NET commands, or via EF, a new history record is created, as shown in **Figure 4**.

The first query shown in **Figure 4** presents the current state of the record with ID=2. The second query, instead, presents the records found in the history table for the same ID. Such an observable state has been determined by two quick updates I made directly in the Management Studio editor. I first changed the column Hour from nine to 13 and then a couple of seconds later I changed the value of the Owner column from Dino to Paul. The first record in the history table says that the originally created record (which I put in the table through EF and a call to SaveChanges) was in a valid state for about five minutes. Then it moved into another state that lasted for a couple of seconds and finally reached the current state. As you can see, the current state isn't stored in the history table. **Figure 5** shows the state of the tables after the record with ID=2 is deleted.

The primary table returns an empty result set when queried for ID=2. The history table, instead, now has a third record whose validity period ranges from the time of the last update to the time of deletion.

Querying for a Specific Entity

Keeping track of all state changes is useful because it doesn't let you miss a thing of what happens in the system. It gives you a complete (and free) log of all database operations. Better yet, it gives you a full list of state changes, which has a much more relevant business role than a plain log of SQL statements. Put another way, conceptually speaking, temporal tables are very close to Event Sourcing; I dare say that temporal tables are a CRUD-based form of Event Sourcing. Let's see how you can query past states of a given aggregate.

Even though the History nested table is useful to figure things out, SQL Server 2016 offers a direct syntax to query temporal data about a given record. Here's the schema of a sample command that retrieve versions of the record with ID=2 in a given time interval:

```
var sql = @"SELECT * FROM Bookings
  FOR SYSTEM_TIME BETWEEN '{0}' AND '{1}'
  WHERE ID=2";
```

A temporal query is a regular query plus the FOR SYSTEM_TIME clause that sets the time interval to consider. The database engine will resolve the query looking at the additional columns in the history table and the content of the primary and nested table. The query is expected to return a list of records. How can you force EF to run a query like this? In EF 6, you can only leverage the SqlQuery method of the DbSet class:

```
using (var db = new EF6Context())
{
  var current = db.Bookings.Single(b => b.Id == 1);
  var time = DateTime.Now.AddMinutes(-5);
  var old = db.Bookings
    .SqlQuery("SELECT * FROM dbo.Bookings
          FOR SYSTEM_TIME AS OF {0} WHERE Id = 1", time)
    .SingleOrDefault();
}
```

Note that for EF 6, the column names returned in the query need to match the property names on the class. This is because SqlQuery doesn't use mappings. If column and property names don't match, then you'd need to alias the columns in the SELECT list, rather than just SELECT *.

With EF Core things are in some respect easier and better. In EF Core, the method to use is FromSql. First off, FromSql method uses mappings; meaning that you don't have to worry about aliasing if column and property names don't match:

```
using (var db = new EFCoreContext())
{
  var current = db.Bookings.Single(b => b.Id == 1);

  var time = DateTime.Now.AddMinutes(-5);
  var old = db.Bookings
    .FromSql("SELECT * FROM dbo.Bookings
              FOR SYSTEM_TIME AS OF {0}", time)
    .SingleOrDefault(b => b.Id == 1);
}
```

Second, you can compose on top of the initial select using LINQ. This means that you can use Where, OrderBy, GroupBy or whatever other LINQ operator and those things in general will be translated to a query of the form:

```
SELECT projection
FROM (SELECT * FROM dbo.Bookings FOR SYSTEM_TIME AS OF {0}) as Bookings
WHERE condition
```

All this said, if you prefer, you could always use plain ADO.NET and data readers to access data stored in temporal tables.

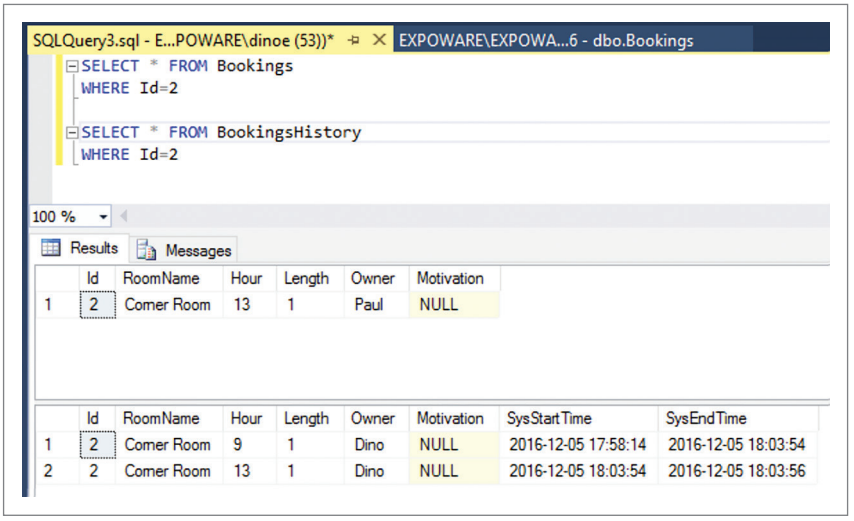


Figure 4 **Updating Records in a Temporal Table**

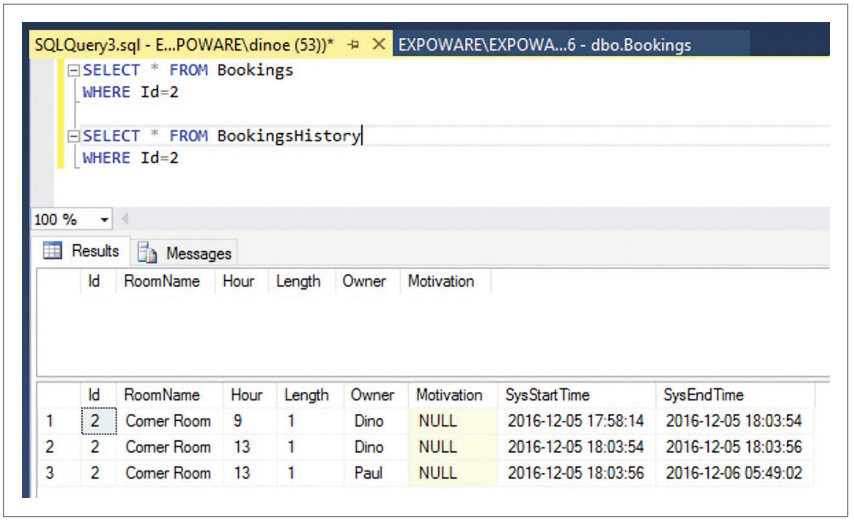


Figure 5 **Deleting Records in a Temporal Table**

Wrapping Up

You can definitely work with temporal tables in a data layer largely based on EF and even if you happen to use plain ADO.NET for the queries, you can leverage LINQ-to-Objects to build complex in memory. The Entity Framework team roadmap shows in the coming months some work items to do on temporal tables. So, let's just wait. ■

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.*

Thanks to the following Microsoft technical expert for reviewing this article: *Rowan Miller*

Data Quality Made Easy.
Your Data, Your Way.

Melissa Data provides the full spectrum of data quality to ensure you have data you can trust.

We profile, standardize, verify, match and enrich global **People Data** – name, address, email & phone.

Our data quality solutions are available on-premises and in the Cloud – fast, easy to use, and powerful developer tools and plugins for the **Microsoft® Product Ecosystem**.

Microsoft SQL Server Integration Services | Microsoft Dynamics CRM | Microsoft .NET | Microsoft Azure | Microsoft Excel

Start Your Free Trial

www.MelissaData.com/msft-pd

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

Hashing Source Code Files with Visual Studio to Assure File Integrity

Mike Lai

The transformation of human-readable code to machine-readable code introduces a challenge to software assurance for all compiled software languages: How does a user have confidence that a software program running on his computer was built from the same source code file created by the developer? That's not necessarily a certainty—even if the source code files are reviewed by subject-matter experts, as they may be in the case of open source software. A critical part of software assurance is trusting that the reviewed source code files are the same source code files that were built into executable files.

During the compilation and linking processes, a set of source code files written in a specific programming language (C#, C++, Objective C, Java and so forth) is transformed into a binary executable file for running on a computer of a specific architecture (x86, x64, ARM, for example). But this transformation may not be deterministic. It's possible that two different sets of source code files could result in two bitwise-identical executable files. Sometimes, this is intentional. Having more or fewer whitespaces or text comments inside the source code files shouldn't affect the binary code emitted by the compiler. On the other hand, it's also possible that a single set of source code files could result in different executable files from different compilation processes. In either case, the problem is one of certainty—knowing for sure that the file you have is the one you want.

To address this issue, it's helpful to use a Visual Studio compiler to hash source code files during compilation. Matching hash values from the compiler to hash values generated from examined source code files verifies that the executable code did indeed result from the particular source code files. Clearly, this is good for users (who would, in fact, benefit further if vendors of other compilers also followed a similar approach). This article describes the new Visual Studio switch for choosing a hashing algorithm, scenarios where such hashes might prove useful and how to use Visual Studio to generate source code hashes.

This article discusses:

- Using the new Visual Studio 2015 switch to choose the SHA-256 hashing algorithm
- Use-case scenarios
- How to use Visual Studio to generate source code hashes

Technologies discussed:

Visual Studio 2015, SHA-256

Generating Strong Hashes During Compilation

A program database (PDB) file is a separate data file that stores the information used to debug a binary executable file. Microsoft recently updated its various compiler file-hashing operations (such as source hashes embedded in PDB files) to use strong cryptographic algorithms.

Native Code Compiler The Visual Studio 2015 native C/C++ compiler, cl.exe, comes with a new switch for choosing a different hash algorithm for the compiler to hash source code files: /ZH:{MD5|SHA_256}. The default is MD5, which is known to be more collision-prone but remains the default because its hash values are computationally cheaper to generate. With the new switch, the compiler implements the SHA-256 option, which is cryptographically stronger than MD5.

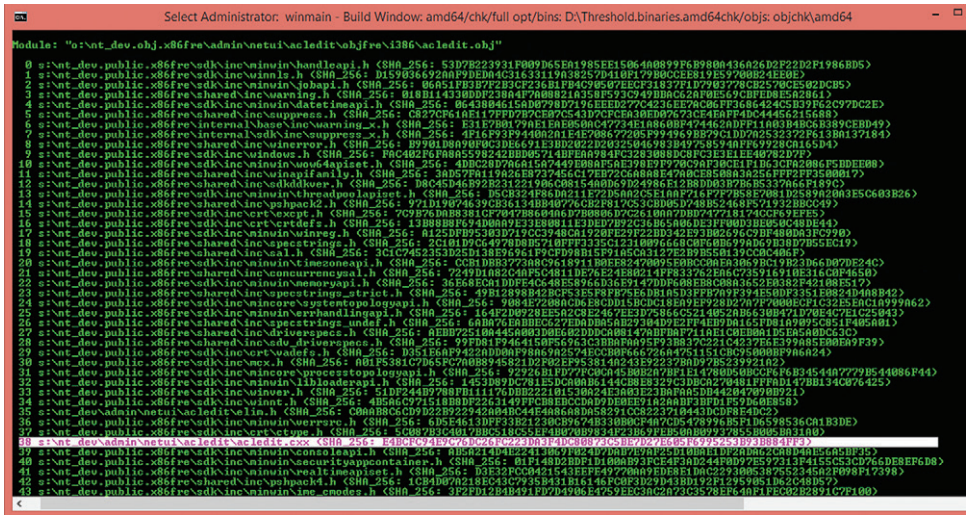


Figure 1 **Using cvdump.exe to View Modules with Their Hashes**

If an SHA-256 hash for a source code file matches an SHA-256 hash stored in the PDB file of a binary executable, it's certain that the same source code file was compiled into the executable, allowing any stakeholder to have confidence in the binary executable file. Effectively, the set of SHA-256 hash values stored in the PDB file of the binary executable file collectively becomes the identifiers in the "birth certificate" of the binary executable file, as these identifiers are registered by the compiler that "gives birth" to the binary executable file.

Using the Debug Interface Access SDK (bit.ly/2gBqKDo), it's easy to create a simple tool such as the Debugging Information Dumper, cvdump.exe (which, along with its source code, is now available at bit.ly/2hAUhyy). You can use the -sf switch of cvdump.exe to view the listing of modules (using their full path names in the local build machine) with their MD5 or SHA-256 hashes, as shown in the command window in **Figure 1**.

When I used a previous version of cvdump.exe to view the same PDB file, I saw the text "0x3" instead of "SHA_256". The 0x3 value is the enum value for SHA_256 and the updated cvdump.exe knows how to interpret it. It's the same enum value that's returned by the IDiaSourceFile::get_checksumType method of the Debug Interface Access SDK.

Managed Code Compiler By default, the Visual Studio 2015 managed code C# compiler, csc.exe, uses the SHA-1 cryptographic algorithm to calculate the source file checksum hash values to store in the PDB files. However, csc.exe now supports a new, optional "/checksumalgorithm" switch to specify the SHA-256 algorithm. To switch to the SHA-256 algorithm, use this option to compile all the C# files in the current directory and place the debugging information, including the source file listing and the SHA-256 hash values, in a PDB file:

```
csc /checksumalgorithm:SHA256 /debug+ *.cs
```

If an SHA-256 hash for a source code file matches an SHA-256 hash stored in the PDB file of a binary executable, it's certain that the same source code file was compiled into the executable.

As part of the .NET Compiler Platform ("Roslyn") open source project, csc.exe is available at github.com/dotnet/roslyn. You'll find support for the SHA-256 source file debug checksum algorithm command-line selector in the file at bit.ly/2hd3rF3.

Visual Studio 2015 csc.exe is compatible only with the Microsoft .NET Framework 4 or higher executable files. The other Visual Studio 2015 .NET Framework compiler used to build executable files prior to version 4 doesn't support the /checksumalgorithm switch.

Managed code PDB files store data differently than native code PDB files. Instead of using the Debug Interface Access SDK, Microsoft DiaSymReader interop interfaces and utilities can be used to read managed code PDB files. Microsoft DiaSymReader is available as a NuGet package from bit.ly/2hrLZJb.

```
<files>
<checkSum="1E, AB, DE, 38, F5, 2D, F5, F7, C3, 3A, 88, AC, DE, 13, 45, C1, F0, A, 5, DD, F9, 51, 6F, 51, 3F, A4, 36, 58, 3A, 71, 8E, B3, "
 checkSumAlgorithmId="8829d00f-11b8-4213-878b-770e8597ac16"
 name="d:\rs1\inetsrv\iis\setup\smplugin\iisplugin\LocResources.Designer.cs"/>

<checkSum="27, 2A, 9C, 7, 9F, AC, DF, 2D, 2B, 7C, 8D, FA, E1, 7A, 3, 97, B8, 2C, 5B, 54, 2, 59, 43, E8, DE, B9, 26, 9D, 39, B2, 65, 73, "
 checkSumAlgorithmId="8829d00f-11b8-4213-878b-770e8597ac16"
 name="d:\rs1\inetsrv\iis\setup\smplugin\iisplugin\NonLocResources.Designer.cs"/>

<checkSum="D8, 63, 54, 80, 41, 4, B1, E3, B9, 9D, 4E, 17, 2, B, 91, C, 5A, DB, 54, 8, 94, B5, 73, 26, 3B, 18, F1, 88, 1B, FF, FA, EC, "
 checkSumAlgorithmId="8829d00f-11b8-4213-878b-770e8597ac16"
 name="d:\rs1\inetsrv\iis\setup\smplugin\iisplugin\BasicPages.cs"/>

<checkSum="18, 52, 9B, E0, 79, 80, 11, 8, F, 71, 35, 1F, 83, D1, 7B, 32, A0, 51, F1, 5A, BF, E2, 96, 79, E, 62, 53, 67, 7B, 0, EC, CD, "
 checkSumAlgorithmId="8829d00f-11b8-4213-878b-770e8597ac16"
 name="d:\rs1\inetsrv\iis\setup\smplugin\iisplugin\TemplatesLoader.cs"/>

<checkSum="16, F5, AD, 4B, 9A, F7, 59, 19, 6D, 77, 5E, 53, 98, 22, 35, AD, FE, 6F, DD, C5, 2C, CD, E5, 48, 4A, A0, 15, 9F, D0, 89, 8F, BD, "
 checkSumAlgorithmId="8829d00f-11b8-4213-878b-770e8597ac16"
 name="d:\rs1\inetsrv\iis\setup\smplugin\iisplugin\UninstallPlugin.cs"/>

<checkSum="D9, F3, 6, B8, 74, A0, 97, AB, ED, 7, 1C, 3C, 3D, 87, E6, B4, A6, 65, 1, B6, 50, F6, D8, 50, 59, 1E, DE, A5, A9, 85, BE, 6F, "
 checkSumAlgorithmId="8829d00f-11b8-4213-878b-770e8597ac16"
 name="d:\rs1\inetsrv\iis\setup\smplugin\iisplugin\WebServerRolePlugin.cs"/>

<checkSum="72, D0, AD, A6, D0, 97, 46, BE, 50, F0, 18, 2E, 81, 6D, 51, 5F, 6C, 36, A7, 79, D0, 5B, 3A, FC, BB, 33, 76, B, 6F, 36, 48, FD, "
 checkSumAlgorithmId="8829d00f-11b8-4213-878b-770e8597ac16"
 name="d:\rs1\inetsrv\iis\setup\smplugin\iisplugin\WelcomePage.cs"/>

</files>
```

Figure 2 **Displaying a Managed Code PDB in XML Format**

The Roslyn project includes a utility called pdb2xml.exe, which you'll find with its sources at bit.ly/2h2h596. This utility displays the content of a PDB in the XML format. For example, the segment in **Figure 2** shows the listing of C# source code files that have been used to compile a managed code executable.

The "8829d00f-11b8-4213-878b-770e8597ac16" GUID in the checkSumAlgorithmId field indicates that the value in the checksum field is an SHA-256 hash value for the file referenced in the name field. This GUID is defined in the Portable PDB Format Specification v0.1 (bit.ly/2hVYfEX).

Compiler Support for SHA-256

The following Visual Studio 2015 compilers support the option for the SHA-256 hashing of source code files:

- cl.exe /ZH:SHA_256
- ml.exe /ZH:SHA_256
- ml64.exe /ZH:SHA_256
- armasm.exe -gh:SHA_256
- armasm64.exe -gh:SHA_256
- csc.exe /checksumalgorithm:SHA256

These compilers are available from the "Developer Command Prompt for VS2015" command window of Visual Studio 2015.

Compilers that don't target Windows platforms don't generally use PDB files for storing their debugging information. These compilers typically produce two executable files simultaneously during a compilation run, one unstripped and the other stripped (bit.ly/2hlfwx6). Full debugging information is stored in the unstripped executable, while the stripped executable doesn't contain any detailed debugging information. The unstripped executable may be suitable for storing the SHA-256 hashes of the processed source code files for the executable. We're planning to reach out to creators of these other compilers to find out which approaches work best for their compilers so that non-Windows-based software using these compilers, such as Office for Android, Office for iOS, or Office for Mac, can get the similar benefits as Windows-based software.

Use-Case Scenarios

Now let's look at some scenarios where the source file hash values can be useful.

> Compilers that don't target Windows platforms don't generally use PDB files for storing their debugging information.

Retrieving Indexed Source Files of a Portable Executable (PE) Binary File The Ssindex.cmd script (bit.ly/2hal0D6) is a utility that builds the list of (indexed) source files checked into source control, along with each file's version information, for storing in the PDB files. If a PDB file has this version control information, you can use the srctool utility (bit.ly/2hs3WXY) with its -h option to display the information. Because the indexed source files also have their hash values embedded in the PDB file, these hash values can be used to verify the authenticity of the source files during their retrieval, as explained in KB article 3195907 (bit.ly/2hs8q0u), ""How To Retrieve Indexed Source Files of a Portable Executable Binary File." Specifically, if the hash values don't match, something may have gone wrong during the generation of the PE/PDB pair or in the source control system. This may warrant further investigation. On the other hand, if the hash values match, it's a strong indication that the retrieved indexed source files were used to compile the PE/PDB pair.

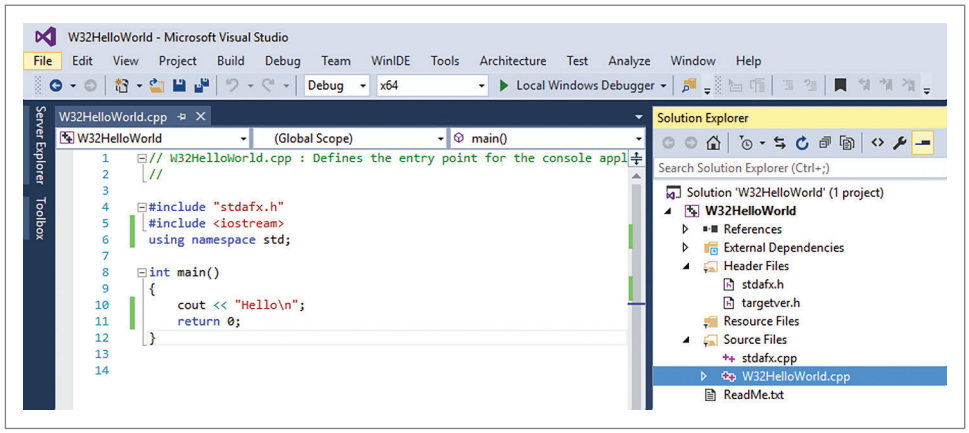


Figure 3 **Win32HelloWorld.cpp**

```
** Module: "c:\Users\documents\visual studio 2015\Projects\W32HelloWorld\W32HelloWorld\x64\Debug\W32HelloWorld.obj"

     0 c:\program files (x86)\microsoft visual studio 14.0\vc\include\xutility (MD5: 267CB213AC0B5C6C21CBC7A7D4682936)
     1 c:\program files (x86)\microsoft visual studio 14.0\vc\include\utility (MD5: F1FB965D91D33DAF49018DD569DB86EC)
     2 c:\program files (x86)\microsoft visual studio 14.0\vc\include\iosfwd (MD5: A34A79668FB37D6889767ACCA1DA97A6)
     3 c:\program files (x86)\microsoft visual studio 14.0\vc\include\xstring (MD5: A1BA4F4B706C8D60CABC08D099469FF0)
     4 c:\program files (x86)\microsoft visual studio 14.0\vc\include\xmemory0 (MD5: 3F7F32C6B92D5CF87B17B2CDEF07F117)
     5 c:\program files (x86)\windows kits\10\include\10.0.10240.0\ucrt\malloc.h (MD5: 928553F88A198C9030B65FA10B6B3DD2)
     6 c:\program files (x86)\windows kits\10\include\10.0.10240.0\ucrt\wchar.h (MD5: AB152FF8E43F2593E1DAFB163B82F9C6)
     7 c:\program files (x86)\windows kits\10\include\10.0.10240.0\ucrt\corecrt_memcpy_s.h (MD5: A5976652B404EDDDBDA326FF9A9488A3)
     8 c:\program files (x86)\microsoft visual studio 14.0\vc\include\limits (MD5: F3EA5C18A23A0A32B4A48A6D682A56E9)
     9 c:\users\documents\visual studio 2015\projects\w32helloworld\w32helloworld\w32helloworld.cpp (MD5: F12DA901F74197DDDE78010EA95F2556)
    10 c:\program files (x86)\microsoft visual studio 14.0\vc\include\istream (MD5: 1D55628EDEA9B69E20C259B7C260FA62)
```

Figure 4 **Cvdump Output Showing Win32HelloWorld.cpp and Its MD5 Hash Value**

Matching Hash Values Produced by a Source File Static Analyzer Today, it's common to use automatic tools to assess the quality of software, as recommended by the Microsoft Security Development Lifecycle (SDL) for the implementation phase (bit.ly/29qEfVd). Specifically, source file static analyzers are used to scan target source code files to assess many different aspects of software quality. These static analyzers typically produce the corresponding real-time results upon scanning the target source code files. As a static analyzer scans individual source code files, it presents an excellent

opportunity to also generate a strong hash (SHA-256) for each of the source code files being scanned. In fact, the Static Analysis Results Interchange Format (SARIF), proposed in the open source project at bit.ly/2ibkbwz, provides specific locations in the static analysis results for a static analyzer to produce the scanned target source code files and their SHA-256 hash values.

Given a PE file, let's assume that the following are available:

1. The compiled source file hash listing from the corresponding PDB file as produced by a compiler.
2. The scanned source file hash listing from the corresponding static analysis result as produced by a static analyzer.

In this scenario, you can review and verify whether the two file hash listings match. If they do, the source files have been scanned for some quality assessment by a static analyzer and you don't need to rescan the source files. Previously, in the absence of file hash listings, you might have had to rescan just to be sure that the static analyzer did a proper assessment.

A Quicker Sanity Check in the Software Update or Hotfix Development Process In situations where you need to release a software update to fix a quality issue found by a source file static analyzer for a released product, the static analyzer should report the absence of that quality issue in the source code files of the pending update. At a minimum, this report would confirm that the update is effective in addressing the original quality issue. In other words, it would validate the intended purpose of the software update. If desired, you or a security reviewer can take the following steps to conduct a quick validation:

1. Confirm that the original static analyzer report identifies the quality issue in question.
2. Confirm that the original static analyzer report includes the hash values of the source files that contain the quality issue.
3. Match the file hash values found in the original static analyzer report with the hash values of the source files of the released product version.
4. Obtain the updated static analyzer report produced from a scan of the source code files of the update using the same static analyzer.
5. Confirm that the previously found quality issue is absent in the static analyzer report for the update.
6. Match the file hash values in the updated static analyzer report with the hash values of the source files of the update.

During these validation steps, you don't need to have access to the actual source code files of either the original released product or the update.

Constructing the Source Code Delta Between Two Versions of Software Reviewing a full set of source code can take time. However, in some cases, when there are changes to the source code, a full review of the source code isn't necessary. As a result, you may be asked only for the source code delta. This request is certainly reasonable as there's no rational basis for a repeat analysis of any portions unchanged since the last review.

> As a static analyzer scans individual source code files, it presents an excellent opportunity to also generate a strong hash (SHA-256) for each of the source code files being scanned.

Previously, without the cryptographically strong hash values of the source code files, it would be difficult to construct the delta subset with any precision. Even if you had a delta subset to offer, the subject matter experts might have had little confidence in your ability to accurately create the delta subset. But that's no longer the case. With the strong cryptographic hash values of the source code files, you can use the following steps to create the delta subset:

1. Obtain the pool, Pool X for example, of hash values of all the source code files for the original product version.
2. Make an exact copy of the file directory, Dir A for example, that contains the source code enlistment of the subsequent product version from which the delta subset will be constructed.

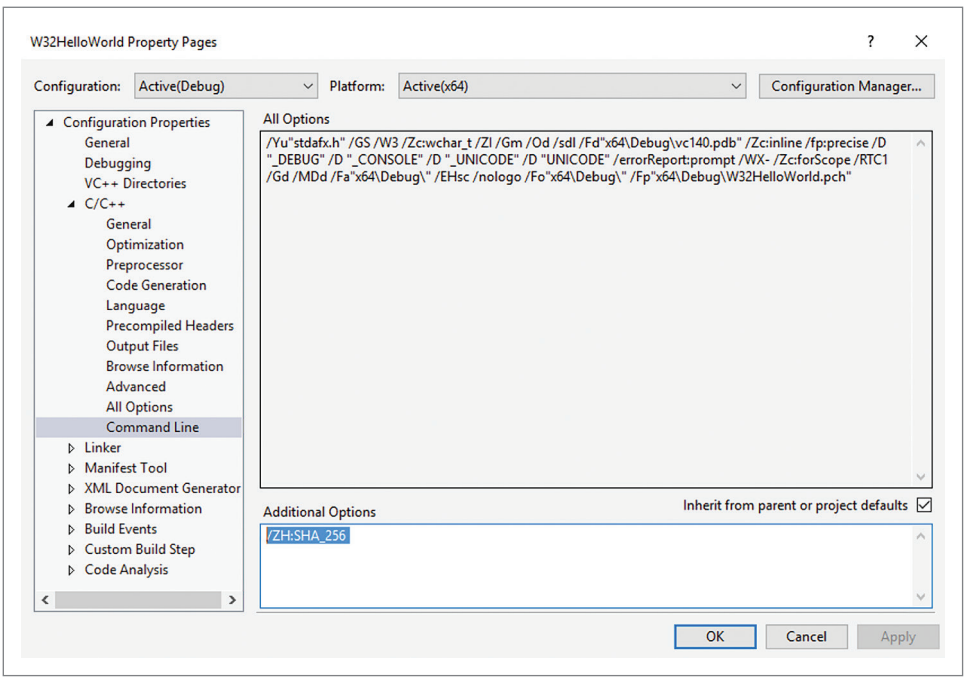


Figure 5 **Switching the Source File Hashing Algorithm to SHA-256**

```
** Module: "c:\Users\documents\visual studio 2015\Projects\W32HelloWorld\W32HelloWorld\x64\Debug\W32HelloWorld.obj"

     0 c:\program files (x86)\microsoft visual studio 14.0\vc\include\xutility (SHA_256: 20C8F017D72895F5B8C251FACC96244B5369BAB0CDC7CB43AE14101BBCB5E6A9)
     1 c:\program files (x86)\microsoft visual studio 14.0\vc\include\utility (SHA_256: E10A2289DE2462367C50F3EAA5E4E4767DF3751F97163154CA39004B6DB43174)
     2 c:\program files (x86)\microsoft visual studio 14.0\vc\include\iosfwd (SHA_256: EBD96FA806C28A89F00D0B1BDAA06FF5730D5A6C2C7A3D182FA8021165F85228)
     3 c:\program files (x86)\microsoft visual studio 14.0\vc\include\xstring (SHA_256: 0BEC7A1D06DE7845E457766311E882A65BB6BD0B32B22AAB952F458F5931B6A7)
     4 c:\program files (x86)\microsoft visual studio 14.0\vc\include\xmemory0 (SHA_256: D3A613BD5F2EC7BA53887299815A63ED3E73A09E9BB161F3A9982797D5BC7426)
     5 c:\program files (x86)\windows kits\10\include\10.0.10240.0\ucrt\malloc.h (SHA_256: 65E88B0B77E83B95A368E98443050235DD02977620CDA42BE5AAB1D49C5DA4A3)
     6 c:\program files (x86)\windows kits\10\include\10.0.10240.0\ucrt\wchar.h (SHA_256: AB152FF8E43F2593E1DAFB163B82F9C6000000000000000000000000000000000)
     7 c:\program files (x86)\windows kits\10\include\10.0.10240.0\ucrt\corecrt_memcpy_s.h (SHA_256: A5976652B404EDDDBDA326FF9A9488A3000000000000000000000000000000000)
     8 c:\program files (x86)\microsoft visual studio 14.0\vc\include\limits (SHA_256: 052A2AD66A7EBE814A4654A4A953F5D4CB16D1E448A55882F6CC2E58C974EDA7)
     9 c:\users\documents\visual studio 2015\projects\w32helloworld\w32helloworld\w32helloworld.cpp (SHA_256: C8BBA0D41B4DA0F52AA79591D41AD2FC13B4A365E60A00EFAA91E6894006C5CD)
    10 c:\program files (x86)\microsoft visual studio 14.0\vc\include\istream (SHA_256: 2FC4A3127A1E24D9C98E810A6EC7753AFB382DE009DF064CA03D25664C80785C)
```

Figure 6 **Cvdump Showing Win32HelloWorld.cpp and Its SHA-256 Hash Value**

3. Prepare a final file folder destination, Dir B for example, to hold only the delta file subset.
4. Sort through all the files in Dir A:
   a. If the hash value of a file matches a hash value in Pool X, do nothing and go to the next file.
   b. If the hash value of a file doesn't match a hash value in Pool X, copy the file to Dir B before moving to the next file.
5. Validate that all the files in Dir B have hash values that match the corresponding hash values of the source files of the subsequent product version.
6. Make the content of Dir B the delta source file subset of the subsequent product version.

Generating the Hash

Now let's take a look at how you do file hashing with the Visual Studio compilers. To do this, I'll use the "Hello, World" application creation example from the online Visual Studio documentation (bit.ly/2haPupF) to:

1. Show where in the output PDB file you can find the hash values of the compiled source files
2. Use the certutil tool (bit.ly/2hlrnPR) to compute the source file hash values to match against the ones found in the PDB file.

To start, I create a new Win32HelloWorld application project in the Visual Studio 2015\Projects folder. In this Win32HelloWorld project, there's only one C++ source file—Win32HelloWorld.cpp—as shown in **Figure 3**.

As you can see, Win32HelloWorld.cpp includes the main function, which displays the "Hello" text.

After making a build for my Win32HelloWorld project, I end up with the W32HelloWorld.exe and W32HelloWorld.pdb files in the Visual Studio 2015\Projects\W32HelloWorld\x64\Debug folder.

The cvdump tool, used with its -sf option against the W32HelloWorld.pdb file, shows the Win32HelloWorld.cpp file and its MD5 hash value in the output shown in **Figure 4**.

The hash value is MD5 because MD5 is the default algorithm for the Visual Studio 2015 compiler, cl.exe. To switch the source file hashing algorithm to SHA-256, I need to supply the /ZH:SHA_256 option to cl.exe. I can do this by adding "/ZH:SHA_256" in the Additional Options box on the Property Pages of the Win32HelloWorld project, as shown in **Figure 5**.

After rebuilding in Visual Studio, I have a new PE/PDB pair of W32HelloWorld.exe and W32HelloWorld.pdb in the Visual Studio 2015\Projects\W32HelloWorld\x64\Debug folder. Now, using the cvdump tool with its -sf option against the new W32HelloWorld.pdb file displays the Win32HelloWorld.cpp file and its SHA-256 hash value in the output, as shown in **Figure 6**.

Now, I can go back to the W32HelloWorld.cpp file in the Visual Studio 2015\Projects\W32HelloWorld\W32HelloWorld folder to check out its SHA-256 hash value. Using the certutil tool with its -hashfile verb against the Win32HelloWorld.cpp file for SHA-256, I get the SHA-256 hash value shown in **Figure 7**.

Clearly, it matches the SHA-256 value recorded in the W32HelloWorld.pdb file. This strongly indicates that the Win32HelloWorld.cpp file was indeed used to compile the W32HelloWorld.exe application, as expected.

For more details on related public tools to work with native code and managed code PE/PDB file pairs, see KB article 3195907, "How To Retrieve Indexed Source Files of a Portable Executable Binary File" (bit.ly/2hs8q0u).

Wrapping Up

I hope this article has shown some potential benefits of a stronger linkage between source code files and the PE file that was compiled with them. You can create a stronger linkage by having the compiler hash the source code files during compilation with the strongest hashing algorithm available—SHA-256. The actual hash values of the source code files produced by the compiler literally become the unique identifiers of the source code files that are used to compile an executable.

Once you understand the value of these unique identifiers, you can use them in different software development lifecycle schemes for tracking, processing and controlling source code files that have a strong linkage to the specific executable files, resulting in a higher level of end-user confidence in the executable files. ■

Mike Lai *just entered his 20th year of Microsoft employment. He is grateful to Microsoft for the various opportunities to contribute to the functionality and the engineering aspects of many of its products. He would like to thank his current management in Trustworthy Computing for their patience to allow his ideas becoming mature and gradual incorporation into the released products, and for their support to participate in Information and Communications Technology security standards organizations.*

```
C:\Users\Documents\Visual Studio 2015\Projects\W32HelloWorld\W32HelloWorld>certutil -hashfile W32HelloWorld.cpp SHA256
SHA256 hash of file W32HelloWorld.cpp:
c8 bb a0 d4 1b 4d a0 f5 2a a7 95 91 d4 1a d2 fc 13 b4 a3 65 e6 0a 00 ef aa 91 e6 89 40 06 c5 cd
CertUtil: -hashfile command completed successfully.
```

Figure 7 **Getting the SHA-256 Hash Value with Certutil**

Thanks to the following Microsoft technical experts for reviewing this article: *Scott Field, Mike Grimm, Sue Hotelling, Ariel Netz, Richard Ward and Roy Williams*

File Format APIs

Working with Files?

CREATE CONVERT PRINT

MODIFY COMBINE

FREE TRIAL

Aspose.Total

Manipulate Word, Excel, PDF, PowerPoint, Outlook and more than 100 other file formats in your applications without installing Microsoft Office.

DOC, XLS, PDF, PPT, MSG, BMP, PNG, XML and many more!

Platforms supported: .NET, Java, Cloud, Android, SharePoint, Reporting Services, and JasperReports

ASPOSE
File Format APIs

CONTACT US

US: +1 903 306 1676
EU: +44 141 628 8900
AU: +61 2 8006 6987

sales@asposeptyltd.com

Try for FREE at
www.aspose.com

Making Bots More Intelligent

Kevin Ashley

I hear it everywhere these days: *bots are the new apps*. One reason is the ease and efficiency they bring to common tasks. Think of it: Instead of jumping from Bing or Google to two or three Web sites or more to book a flight or make reservations at a restaurant, you're able to simply ask some kind of digital assistant to do it for you. But there may be an even more significant reason for app developers to embrace this trend: Mobile apps are becoming more and more expensive to create. Building just one mobile app takes anywhere between 30,000 and half a million dollars for each platform: iOS, Android and Windows.

True, there are some cross-platform technologies—such as Unity or Xamarin—that can help mitigate the cost of developing for multiple platforms. But bots are emerging as another option, shifting the paradigm by employing a conversational interface. This approach leverages communications used by humanity for thousands of years and spans voice and text (think Skype or Messenger). The Conversational UI (CUI) can also be enhanced with images, videos and traditional controls such as Buttons and CardActions. In bots, the focus is on simplicity and the natural aspects of input, rather than crafting the UI with traditional controls.

Microsoft Bot Framework and Azure Bot Service are in preview. All information is subject to change.

This article discusses:

- The benefits of bots
- Language intelligence with LUIS
- The Bing location control
- Azure Bot Service and Azure Functions

Technologies discussed:

Microsoft Bot Framework, LUIS, Azure Bot Service, Azure Functions

> Bots can address the challenges of Web sites that are becoming increasingly difficult to navigate, with many using old or outdated Web frameworks or standards.

The benefit? A potential aggregate savings in the billions of dollars as organizations move away from building distinct UIs for each screen paradigm and instead adopt conversational interfaces that work across all devices and form factors, with a screen or without. For example, I built an Active Fitness bot that helps you find running, cycling, hiking, skiing and other trails near you. (Visit bit.ly/2knZVbr to add to Skype.) While the app features a rich UI and controls, the bot is designed to understand your conversational input, as shown in **Figure 1**.

The impact extends beyond app development. Bots can address the challenges of Web sites that are becoming increasingly difficult to navigate, with many using old or outdated Web frameworks or standards. Try to complete a task at any customer service Web site, for example, and you'll probably spend a good half hour trying to navigate the Web site menu.

And then there's the unique reach of bots. Imagine a card game app, such as Solitaire. Before bots, I had to build a version for each platform. Now, however, bots disrupt the trap that is platform lockdown. Suddenly, my Windows-based apps work as bots on all my friends' iPhones and Androids. Why? Because bots work in channels—Skype, Telegram, Facebook Messenger, Slack and other ubiquitous app platforms.

I figured it'd be cool to build the most popular game in the world using Microsoft Bot Framework, so **Figure 2** shows my take on Solitaire, all ready to play. (Visit bit.ly/2jrzP7S to add to Skype.)

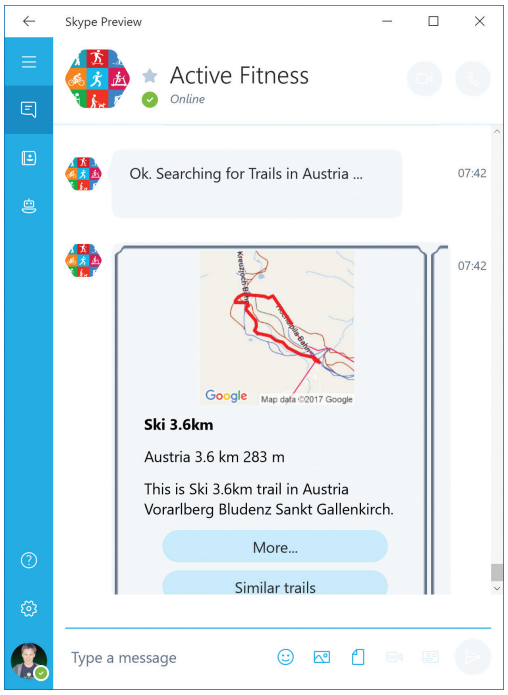


Figure 1 **Active Fitness Bot That Shows You Trails**

To start creating bots, go to the Microsoft Bot Framework site (dev.botframework.com). You can build bots using the Microsoft .NET Framework or Node.js, whichever technology you prefer. Recently, Microsoft added Azure Bot Service (bit.ly/2knEtU6) as yet another convenient way to implement bots. Azure Bot Service provides a browser-based experience for building bots from easy-to-use templates. The service uses Azure Functions, an event-based serverless code architecture with "functions" that allow you to save money and scale your bots better.

Here's what's great about Azure Functions: When I started building bots, I realized that each time I needed a bot I'd have to create a new Web app to provide an app container. That's fine for one app, but not for multiple bots, which would each require a separate infrastructure. Instead, Azure Functions allows you to host small pieces of code in the cloud, without worrying about the whole application infrastructure.

Intelligence and Bot Development

Just like apps, bots are here to help you with a specific task they can do best. For example, my Solitaire bot can play Solitaire with you and maybe even teach you rules and tricks of the game. True, it's not likely to pass the Turing test (a way to measure a machine's intelligence developed by Alan Turing in the 1950s), but that would probably be overkill for most bots.

Likewise, a bot for your local motor vehicle office should know certain things about submitting a payment for your license registration or how to help you pay a ticket or submit a driver license extension. A weather bot needs to answer weather-related questions, and the intelligence of the bot might include ways to figure out location and user needs for specific weather data.

Bots can automate many things via a conversational interface, but they can also be enhanced with images, buttons and other controls that make interaction even more convenient with mobile devices. Bot intelligence really comes into play when conversations become open-ended. Using dialogs for a predefined set of answers is a standard, relatively easy programming task. Responding to a human conversation, on the other hand, implies intelligence. Fortunately, Microsoft Bot Framework allows you to do both: LuisDialogs enables support for natural language-based conversations, while more traditional controls let users press buttons, carousels or other controls to select options.

Intelligence is at the heart of the conversational UI employed by many bots, enabling intuitive interaction with human users. The growing number of Cognitive Services from Microsoft (bit.ly/2jx1kMQ) allows you to plug intelligent capabilities right into your bot. These services include (but are not limited to):

- Vision
- Speech
- Language
- Knowledge
- Search
- Location

Check out Alessandro Del Sole's fantastic article on Cognitive Services at msdn.com/magazine/mt742868.

Making your bot more intelligent involves the use of some common measures that might not make it through a Turing test, but will certainly improve conversations with the bot:

- Dialogs, images and other ways to gather normalized data, convey a message and simplify conversations
- Localization

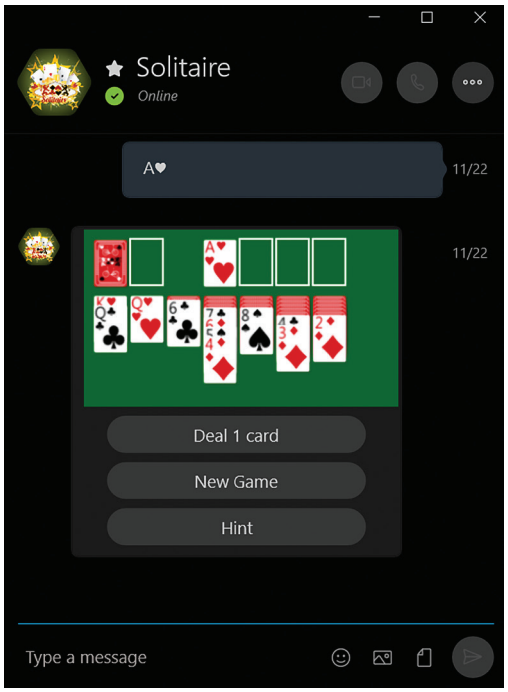


Figure 2 **My Solitaire Bot on Skype Handles 500,000 Requests per Week**

- Help and dialog pipelines to improve basic bot functions
- User data (such as location) to eliminate duplicating questions

You don't want to be too ambitious and try to create a bot that resembles Douglas Adams' city-sized Deep Thought computer (from "The Hitchhiker's Guide to the Galaxy"). Rather, when I started building my bots, I limited them to functions that were essential to primary tasks—like playing a card game, or suggesting best trails to run—and then enhanced them with intelligence.

Language Intelligence with LUIS

The first intelligence component to consider adding to a conversational interface is Microsoft Language Understanding Intelligence Service (LUIS). This service works by analyzing conversations or sentences you send to it, and extracting actual *intents* and *entities* specific to your application. To add a LUIS dialog to your bot, simply create a class that inherits LuisDialog, register your LUIS app at luis.ai and provide app id and key to your dialog:

```
[LuisModel("<YOUR_LUIS_APP_ID>", "<YOUR_LUIS_APP_KEY>")]
[Serializable]
public class IntelligentLanguageDialog :
LuisDialog<object>
```

For more in-depth LUIS guidance, please check out Ashish Sahu's January 20176 *MSDN Magazine* article at msdn.com/magazine/mt745095.

In my Active Fitness Trails bot (**Figure 3**) I defined intents related to fitness activities, and I expect the user to provide location and fitness activity as part of an intelligent bot conversation. I want a user to be able to say, for example, "Show me ski trails in Colorado," and the bot should understand this and come up with a set of trails. Users of my Active Fitness app run millions of trails daily, and by querying Active Fitness they can map the GetActivityLocation intent from my LUIS model, which returns geography and activity entities back to the bot.

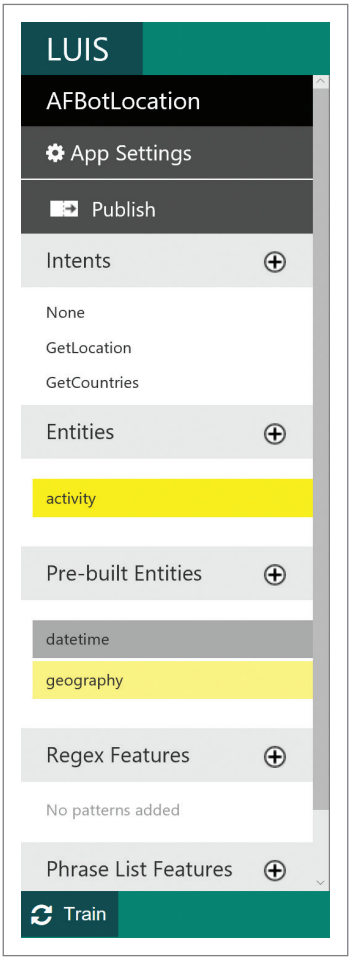


Figure 3 **Model in LUIS**

Imagine how much coding I'd have to do to extract all that data from non-normalized user input! Even a short human sentence often involves a level of complexity that only machine learning tools can understand and process. Add all possible variations of that simple sentence and you realize that bot intelligence is a non-trivial task. Fortunately, LUIS acts as an intelligent conversation analyzer and extractor that does all the hard work for you. All you need is to add it to your bot.

> Even a short human sentence often involves a level of complexity that only machine learning tools can understand and process.

Once the model is defined, I can train it by providing utterances that help LUIS understand what my intents are and how to get entities for my bot. Training the model for my GetActivityLocation intent, I provided the following utterances: "Show cycling trails nearby," "Show latest snowboard trails in Austria," and, "Where are the longest ski trails in Utah?" As **Figure 4** shows, when LUIS processes entities, it highlights entities I specified in my model. What if LUIS doesn't recognize an entity? This is where you train it by manually mapping entities. For example, I had to train my bot to recognize "snowboard" as an activity.

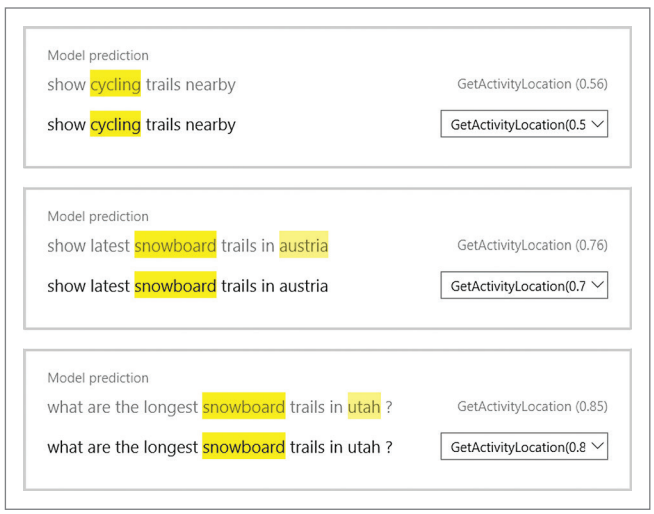


Figure 4 **Model Training in LUIS**

Figure 5 **Handling Intents**

```
[LuisIntent("None")]
  public async Task NoneIntent(IDialogContext context, LuisResult result)
  {
    await context.PostAsync(
      $"You have reached the none intent. You said: {result.Query}"); //
    context.Wait(MessageReceived);
  }

  // Go to https://luis.ai and create a new intent,
  // then train/publish your luis app.
  // Finally, replace "GetActivityLocation" with the name of your newly
  // created intent in the following handler.
  [LuisIntent("GetActivityLocation")]
  public async Task GetActivityLocation(IDialogContext context, LuisResult result)
  {
    await context.PostAsync(
      $"You have reached the GetActivityLocation intent. You said:
      {result.Query}"); //
    context.Wait(MessageReceived);
  }
```

FASTEST WAY TO BUILD HIGH-PERFORMING, FEATURE COMPLETE APPLICATIONS

Easy-to-use UI controls trusted by developers and enterprises worldwide

Windows, Web, and Mobile UI Controls, Reporting, and Productivity Tools

Global Xamarin Dev Days Partner

Visual Studio 2017 RC
RC Extension Sim-Ship Partners

- Extensible controls with easy-to-use universal API
- Industry's best high-performance grids, charts, and reports
- Small footprint reduces app bloat
- Global support and community

ComponentOne Studio

Download your free 30-day trial at
www.ComponentOne.com

GrapeCity

Figure 6 **Finding the Country**

```
// Go to https://luis.ai and create a new intent, then train/publish your luis app.
  // Finally, replace "GetActivityLocation" with the name of your
  // newly created intent in the following handler.
  [LuisIntent("GetActivityLocation")]
  public async Task GetActivityLocation(IDialogContext context, LuisResult result)
  {
    await context.PostAsync(
      $"You have reached the GetActivityLocation intent. You said:
      {result.Query}"); //
    EntityRecommendation country;
    if(result.TryFindEntity("builtin.geography.country", out country))
    {
      await context.PostAsync($"Country: {country.Entity}");
    }

    context.Wait(MessageReceived);
  }
```

Next, you define methods that handle intents returned by your LUIS model. You can have methods that handle a None intent or any other intents from your model, as **Figure 5** shows.

> Fortunately, the Microsoft Bing location control is now part of Bot Framework, which makes gathering location-based data much easier for developers.

Each time the model determines that the input matches a GetActivityLocation intent, it'll call my bot method. The result is returned by LUIS in the LuisResult object and includes entities such as location and activity. For example, I want to check if the entities contain a country. To do this I use the TryFindEntity method of the LuisResult object, looking for the "builtin.geography.country" type of entity. LUIS provides a number of pre-built entities, which greatly simplifies the job for you. You can find a complete list of these entities at bit.ly/2kWgCHR.

Notice that the builtin.geography entity contains sub-entities, such as country and city. An entity that contains sub-entities is called a composite entity. In this case, I'm interested specifically in the country sub-entity, as shown in **Figure 6**.

Of course, I also need to know the activity entity (you can use Active Fitness to track more than 50 activities, such as running, cycling, skiing and snowboarding). So, when I ask: "Where's the best skiing in Austria?" I expect the Active Fitness bot to return "skiing" as my activity entity, as shown in **Figure 7**.

Consequently, I updated my GetActivityLocation method to return the activity. Notice that unlike builtin.geography.country, my activity is not a built-in entity. Still, LUIS figures it out pretty well once I provide a few utterances while training the model, as shown in **Figure 8**.

Bot Framework provides in-depth generic code examples for C# at bit.ly/2gHupjg, and for Node.js at bit.ly/2kWolWx.

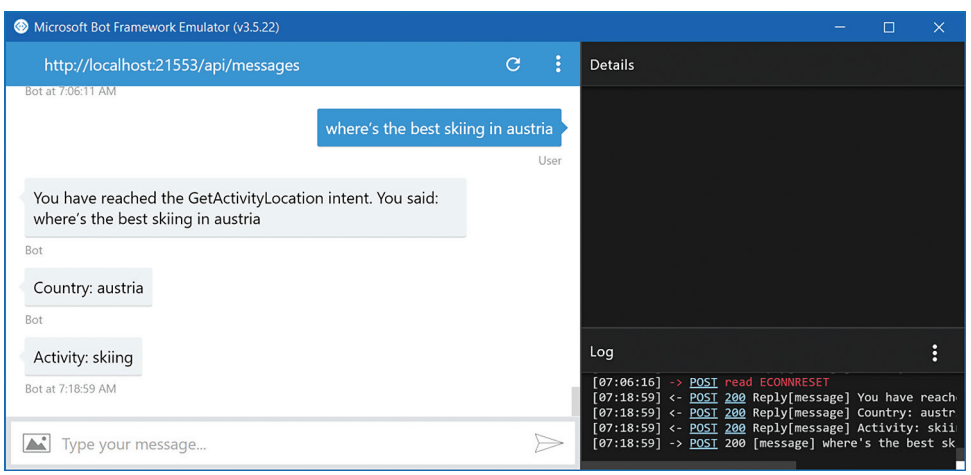


Figure 7 **The GetActivityLocation Intent in action with Bot Emulator**

Practical Bot Tips

Microsoft is pushing hard on the bot development front, releasing tools like the Language Intelligence Service (LUIS) and Azure Bot Service to help streamline the creation and management of this new class of software. The good news is, most apps today can be easily transformed to bots, with the conversational UI (CUI) a great fit for multiple tasks and types of apps. To get the most out of your bot development effort, here are a few things to consider:

Play for Reach: One of the most compelling things about bots is they run in channels that are found on every PC, tablet and smartphone—like Skype, Facebook Messenger and Telegram. Maximize audience reach by publishing bots to multiple channels. Also, take advantage of available services to localize your bot to multiple languages.

Be a Conversationalist: Make good use of the conversational interface, which is a great fit for multiple tasks and types of apps. For best results, focus on conversational interfaces like chat, speech and language, rather than traditional UI controls like buttons.

Service, Please: The new Azure Bot Service is a cloud-based platform that minimizes infrastructure overhead and allows your bot projects to scale. Behind this capability is Azure Functions, which enables developers to host small pieces of code in the cloud.

Get Focused: When building bots, focus on the specific task or tasks each bot needs to solve. Don't overcomplicate—more AI isn't always better. Start with the primary task your bot needs to solve, then inject intelligence.

Speak Up: Make use of the smarts built into LUIS to enable open conversations and provide input for your bot. For best results, take care to train your LUIS model, focusing on a concise set of entities and intents. The less "fuzziness" in your model, the better the AI will perform!

Get news from MSDN in your inbox!

Sign up to receive **MSDN FLASH**, which delivers the latest resources, SDKs, downloads, partner offers, security news, and updates on national and local developer events.

msdn magazine

msdn.microsoft.com/flashnewsletter

Figure 8 **Training the Model with Utterances**

```
// Go to https://luis.ai and create a new intent, then train/publish your luis app.
  // Finally, replace "GetActivityLocation" with the name of your newly
  // created intent in the following handler.
  [LuisIntent("GetActivityLocation")]
  public async Task GetActivityLocation(IDialogContext context, LuisResult result)
  {
    await context.PostAsync(
      $"You have reached the GetActivityLocation intent. You said:
      {result.Query}"); //
    EntityRecommendation country;
    if(result.TryFindEntity("builtin.geography.country", out country))
    {
      await context.PostAsync($"Country: {country.Entity}");
    }
    EntityRecommendation activity;
    if (result.TryFindEntity("activity", out activity))
    {
      await context.PostAsync($"Activity: {activity.Entity}");
    }

    context.Wait(MessageReceived);
  }
```

Get Smart About Location

Many bots provide location-based responses, but coding around location responses can get tedious. In the previous example, I used LUIS intelligence to get a built-in geography entity parsed by LUIS from the conversation. Fortunately, the Microsoft Bing location control is now part of Bot Framework, which makes gathering location-based data much easier for developers.

> It's worth mentioning that bots are also open to third-party APIs and services.

Location control for Bot Framework also provides a visual interface that includes maps (see **Figure 9**). This can be very convenient if your bot needs to provide addresses that include ZIP codes, city, region and locality data from an easy-to-use or native interface that works in channels such as Skype and Facebook Messenger.

To start using a Bing Location control, simply obtain a Bing Maps API key and a Location dialog component for either a .NET project (via NuGet) or Node.js (via npm), then initialize and call an instance of the Location dialog in your code:

```
var options = LocationOptions.UseNativeControl |
LocationOptions.ReverseGeocode;
var requiredFields = LocationRequiredFields.Locality |
                     LocationRequiredFields.Region |
                     LocationRequiredFields.Country;
var prompt = "Where are you looking for trails?";
var locationDialog = new LocationDialog(
  apiKey, this.channelId, prompt, options, requiredFields);
context.Call(locationDialog, this.
ResumeAfterLocationDialogAsync);
```

LocationDialog provides options for requesting various address fields and customizing prompts. When the result is returned, you can handle it in a resume method, as shown in **Figure 10**.

Meet My Bots

I've developed quite a few bots over the last few months, so I decided to share them with you from one convenient location, listed in **Figure A**. Check them out and see if they might inspire you to create your own. (The link provided with each adds that bot to Skype.)

Figure A **My Collection of Bots**

Bot	Description
Solitaire (bit.ly/2jrzP7S)	Solitaire is considered the world's most popular card game. Now instead of launching an app, you can use your favorite chat client, like Skype, to enjoy the game.
Active Fitness (bit.ly/2knZVbr)	Find trails around the world—running, cycling, walking or skiing–all from world's top fitness social network.
UNO (bit.ly/2k4AzyH)	Play this world-famous card game on Skype, Messenger or Telegram.
Freecell (bit.ly/2l0NM9b)	Freecell is a great and challenging card game with multiple levels of difficulty. Start with easy levels and learn to play.
Crazy Eights (bit.ly/2kY7EdN)	Check out this easy-to-learn card game.
Card Games Chest (bit.ly/2klXOCV)	Learn to master some of the most popular card games, including Solitaire, UNO, Crazy Eights, 101, Freecell and Mau-Mau.

Beyond LUIS

In addition to LUIS, Cognitive Services provides a number of other APIs that can empower your bots with skills that may seem incomprehensibly complex for an individual developer or business to implement. Luckily, you don't have to do it on your own:

- **Recommendations API** allows you to include recommendations on products (such as products frequently bought together) or personalized user recommendations.
- **Vision API** adds advanced image and video skills to your bots to recognize objects, people's faces, age, gender or even feelings.

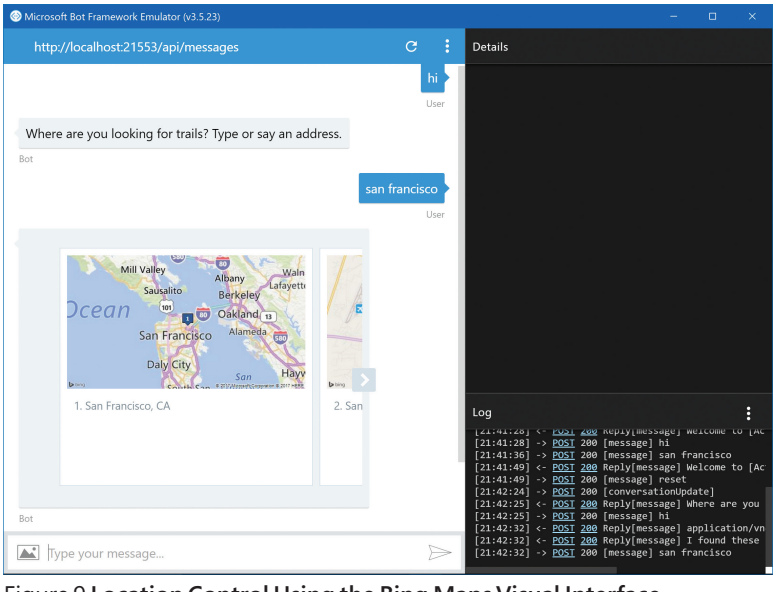


Figure 9 **Location Control Using the Bing Maps Visual Interface**

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

Aspose.Total for .NET | from $2,939.02

ASPOSE File Format APIs

Every Aspose .NET API 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

DevExpress DXperience 16.2 | from $1,439.99

DevExpress

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

- WinForms - New TreeMap control, Chart series types and Unbound Data Source
- WPF - New Wizard control and Data Grid scrollbar annotations
- ASP.NET - New Vertical Grid control, additional Themes, Rich Editor Spell Checking and more
- Windows 10 Apps - New Hamburger Sub Menus, Splash Screen and Context Toolbar controls
- CodeRush - New debug visualizer expression map and code analysis diagnostics

ActiveReports 11 | from $1,567.02

GrapeCity

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

- Visual Studio-integrated report designer
- Extensible optional report server with built-in scalability
- Responsive HTML5 Report Portal
- Royalty-free redistribution
- Source data from JSON files, Web services and REST API using the built in JSON data provider

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
7F Kojimachi Ichihara Bldg
1-1-8 Hirakawa-cho
Chiyoda-ku
Tokyo, 102-0093
Japan

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

GSA Schedule Contract GS-35F-0188R

Figure 10 **Handling Location Results in a Resume Method**

```
private async Task ResumeAfterLocationDialogAsync(
  IDialogContext context, IAwaitable<Place> result)
  {
    var place = await result;

    if (place != null)
    {
      var address = place.GetPostalAddress();
      var formattedAddress = string.Join(", ", new[]
      {
         address.Locality,
         address.Region,
         address.Country
      }.Where(x => !string.IsNullOrEmpty(x)));

      await context.PostAsync(
        "Where are you looking for trails " + formattedAddress);
    }

    context.Done<string>(null);
  }
```

- **Academic Knowledge API** can add academic knowledge, create Q&As, and add specific knowledgebase skills to your bot.

It's worth mentioning that bots are also open to third-party APIs and services. You can use your own services and APIs to make your bots unique and solve problems that haven't been solved before. Indeed, Bot Framework brings a new world of opportunities, as well as human knowledge evolution and accessibility, via simple conversational interfaces and easy-to-use APIs.

New and Exciting

Besides regular updates to Bot Framework for the .NET Framework and Node.js, as well as REST APIs, Microsoft recently added features that take bot development to the next level.

One of the latest additions to Bot Framework, Azure Bot Service, now offers an easy way to leverage the convenience and scalability of the cloud for your bot projects (bit.ly/2knEtU6). Bot Service, shown in **Figure 11**, uses Azure Functions and a collection of quick-start templates to help you get the most out of your bot code and allow your bots to scale to any level (bit.ly/2kuo9Bb). For example, you can choose from Basic, Form, Proactive, LUIS, or Question and Answer bot templates, each providing a ready-to-run bot you can expand as needed.

Azure Functions is the technology behind Azure Bot Service, providing event-driven experiences based on serverless architecture and compute-on-demand. What this really means is that you no longer need an expensive app-hosting infrastructure for each bot. Instead, you can make bots much more lightweight, scale them more easily and save on development effort and time, all while simplifying connectivity with Cognitive Services.

QnA Maker (qnamaker.ai) is another addition worth mentioning. Available from Azure Bot Service as a template bot project, QnA Maker fills a very common scenario for bots that answer typical questions from an existing knowledgebase.

Wrapping Up

Microsoft Bot Framework offers a quick and easy way to start building bots, providing a conversational interface for new and existing apps or services. It lets you create new experiences and reach billions of users connected through Skype, Facebook Messenger, Telegram and other channels. Microsoft Cognitive Services includes many intelligent APIs you can add to your bot.

Bot Framework offers a quick and easy way to start building bots, providing a conversational interface for new and existing apps or services.

The Bot Framework ecosystem is growing rapidly, and you can take advantage of this by adding services included with Cognitive Services or using APIs from third-party providers to make your bots smarter and more capable. The journey that begins with a very simple bot can lead to many discoveries for you and your users. Bots are indeed the new apps (or extensions of existing apps), and with intelligence added, they truly become very powerful. Moreover, they reduce your development time, effort and cost. ■

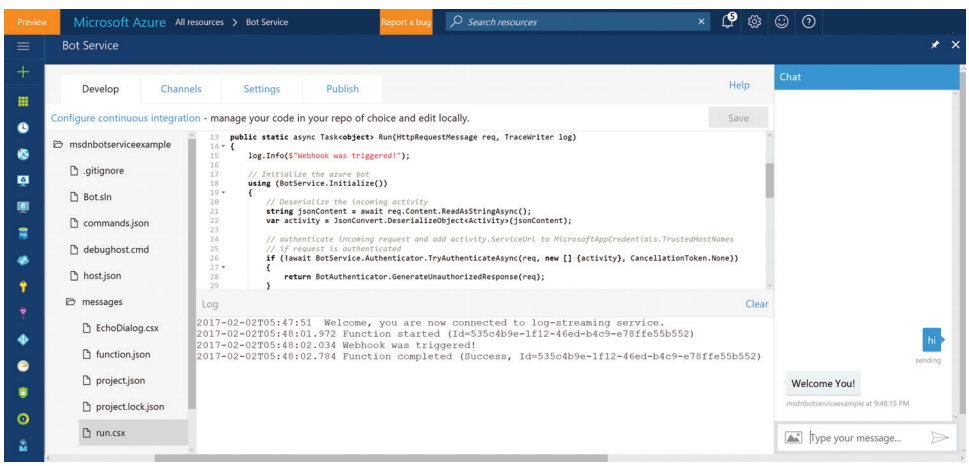


Figure 11 **Editing Bot Service Code Directly in the Browser**

Kevin Ashley *(@kashleytwit) is an architect evangelist for Microsoft. He's coauthor of "Professional Windows 8 Programming" (Wrox, 2012) and a developer of top apps, bots and games, most notably Active Fitness (activefitness.co). He often presents on technology at various events, industry shows and Webcasts. In his role, he works with startups and partners, advising on software design, business and technology strategy, architecture, and development. Follow his blog at kevinashley.com and on Twitter: @kashleytwit.*

Thanks to the following Microsoft technical expert for reviewing this article: *Mat Velloso*

Active Events: One Design Pattern Instead of a Dozen

Thomas Hansen

For as long as there's been software, there's been software that fails. According to studies, 25 percent of all projects fail completely, while 50 percent can be described as "challenged." Imagine if a car manufacturer had the same statistics, and 25 percent of their cars wouldn't start, or exploded when you turned on the lights, while 50 percent could do only 25 miles per hour or used 15 gallons of gasoline per mile. More important, what if customers wouldn't even know which cars would work before they'd paid for theirs. This is pretty much the state of software development today.

Various solutions have been proposed to improve this state of affairs, none of which goes to the root of its cause—the code! In my opinion, the reason software projects fail can generally be reduced to the same problem: spaghetti code. And the reason the code is in such a sorry state is because there's really no way to accommodate change. That's why methodologies such as eXtreme Programming and Agile software development have been recommended. However, none of these can actually help create better code. Being Agile doesn't magically help you avoid spaghetti code. To become Agile, what you need is to be able to separate your concerns into atomic building blocks.

What I'm presenting here isn't an alternative to Agile or other such methodologies. Agile is, after all, a management framework, not an architectural design pattern. Instead, I'm proposing a design methodology that allows you to more easily become Agile because you can prepare up front for change. Over the past few years I've developed a design pattern I call "Active Events." This design pattern lets you create small, reusable components that you can easily extend, modify, reuse and exchange with other parts. It allows you to think differently about software so you can "orchestrate" your code, almost as if the individual pieces were made of Legos. You may be thinking that this is pretty much what writing code using an object-oriented programming (OOP) paradigm is. However, Active Events is very much different than traditional OOP.

This article discusses:

- The limitations of object-oriented programming
- How Active Events work
- Polymorphism and encapsulation
- The Node.cs data class

Technologies discussed:

Active Events, Object-Oriented Programming

Editor's Note

When I approached *MSDN Magazine* Senior Contributing Editor James McCaffrey to review the preliminary draft of this article, he came away fairly outraged by some of the opinions and ideas put forward by the author. Most days, that would spell curtains for an article draft. But as McCaffrey notes, it's all too common for new ideas in software engineering to be dismissed simply because they're new. And while McCaffrey says he's still outraged by many of the statements in this article, we both agree that it may provoke a lot of thought about software design and methodology paradigms. *—Michael Desmond*

Check Your Own Code

Do me a favor: Open up the last solution you built in Visual Studio and check out your dependency graph. How many projects and classes can be extracted from your solution? How many classes could be reused in future projects? How many could you replace with other classes without breaking your software?

My guess is that your project and class dependencies look like they do in most other projects. It's highly likely your project's dependency graph resembles spaghetti. Sorry for being brutally honest, but that's what my experiences indicate. But, relax, you're not alone. Although I don't have hard data, I would claim 98 percent of the world's code suffers from the same problem. Besides, I wouldn't force you to confront this issue unless I had a solution for you.

If you could somehow untangle those dependencies and create reusable components, such that none of the individual parts of your code reference any other parts, your programming life would improve substantially. If your projects and classes were not as entangled as they are today, you could more easily exchange individual pieces. If your classes were untangled, you could reuse them in future projects. If your projects held minimal dependencies, you'd have a range of components that are more likely reusable instead of a monolithic monster. Active Events facilitates zero dependencies across your classes and projects. When it comes to software entanglement, zero is the only acceptable number!

How Do Active Events Work?

The Active Events design pattern is actually surprisingly easy to understand. Instead of creating interfaces to separate the implementation and consumer of a piece of logic, you simply mark your methods with an attribute. Here's an example:

```
[ActiveEvent (Name = "foo.bar")]
protected static void foo_bar(ApplicationContext context, ActiveEventArgs e)
{
  var input = e.Args["input"].Value as string;
  e.Args.Add("output", "Hello there " + var);
}
```

This syntax is taken from Phosphorus Five, an open source Active Event implementation I created. See "Software That Almost Doesn't Exist" for more information.

To invoke a method that has been defined using the Active Event paradigm is equally simple. Instead of invoking your method directly, you indirectly invoke it through its Active Event name:

```
/* ... some method in another assembly ... */
var args = new Node ();
args.Add ("input", "John Doe");

/* context is your ApplicationContext instance, which keeps
   track of your Active Events */
var result = context.Raise ("foo.bar", args)["output"].Value as string;
```

The Raise invocation invokes the foo.bar Active Event. Note that there are zero dependencies between where you invoke your Active Event and its implementation. In addition, there are no common classes shared between them. Only data is passed in and out of the events. Also notice that all Active Events have the exact same signature. Hence, classes, interfaces, and methods move a bit behind the scene and become less a focus in the source code.

The loading of the Active Event components is completely dynamic and can be done either through a list of assemblies referenced in your application's configuration file or by automatically loading all DLLs from a folder and registering them as Active Event handlers. Internally the Active Event "kernel" will traverse all types in your DLL during loading and store a MethodInfo reference to its Active Event methods in a string/MethodInfo dictionary. This dictionary is later used during invocation of your event. If you wish to scope an Active Event or associate state with it, you can dynamically add an instance of a class as an Active Event listener and use an instance method instead of a static method as your Active Event sink, which is what's being done in, for instance, the plugins/p5.web project in Phosphorus Five.

> The Active Events design pattern is actually surprisingly easy to understand.

This creates an environment where your different parts are simple building blocks, invoking each other indirectly, which allows you to easily exchange any single Active Event assembly with another implementation. Completely changing the resulting application is as easy as changing a configuration setting or replacing one DLL with another. And it's just as easy to, if you wish, change the string "foo.bar" to "foo.bar-2." At the same time, each assembly in your solution can still invoke methods in your other assemblies, without sharing so much as a plain old data (POD) structure with the other party. You basically have a constant "yet another layer of abstraction" at your disposal. Or put another way, your components have been atomized.

Let's compare this to what you'd have to do in traditional OOP. First, assuming you're using a design-by-interface approach, you'd have an interface and at least one implementation of the interface. Depending on scenario, you might want to use an abstract factory scheme for creating objects. So that makes two classes and an interface, which could easily total 30 or more lines of code. In addition, if you wanted to create a true plug-in from your implementation, you'd end up with one project for the consumer parts, which consumes your plug-in, and one project for the implementation parts—the project with your class that implements the interface. And if you were to go with a completely modular approach, you might use a third project to contain your interface and possibly your abstract factory. Then, as you looked at the result, you'd realize you ended up with at least three cross-component references among your three projects. Moreover, you can't add to or change the input or output parts of your interface without having to modify all three projects. Compare that to the one-line solution, with no references at all, that the Active Event example gives you, which doesn't even care what input or output arguments you supply to it. The traditional approach could well have twice as many lines of code, or even more, resulting, in my experience, in a far more complex dependency graph, far more rigid result, and far more complex code.

OOP Is Not OO

I love OOP. It's a beautiful paradigm that easily allows you to encapsulate code logic, methods and data. However, one can argue that

OOP hasn't fully solved all coding issues. In some situations when using OOP you must define multiple classes, which are too often closely coupled, and create dependencies.

In OOP, you need to know a method's signature; you need to have access to an interface; and you need to know how to create an instance of this interface. In addition, you often need to share types between the consumer and the implementation of your interface. This creates dependencies among your building blocks. Using OOP, it isn't uncommon to have to create multiple classes just to create a simple "Hello World" plug-in. If you change an OOP interface, unless you've designed your interface very carefully, you may need to recompile multiple assemblies. You can end up with much of your code being abstract interfaces and boilerplate, or just having to accept the fact that a change in an interface may have a domino effect and require significant additional code changes. Whenever something is excessively complex or requires a lot of tedious, repetitive work, your gut should tell you that it's wrong.

Ask yourself this question: "If OOP provides the ability to create object-oriented (OO) software, why is it so difficult to use it successfully?" In my opinion, OOP isn't equivalent to OO. If it were, you wouldn't need to know so many design patterns to fully take advantage of it.

I believe OOP, as currently practiced, has several fundamental limitations, and all those design patterns you need to implement to adequately use OOP are but a symptom of a much more fundamental problem in its architecture: the inability to deliver OO software. Active Events fix these problems, not by implementing better abstract factories or new class constructs, but rather by changing the very way methods and functions are invoked, completely taking OOP out of the equation. To create OO software, what's needed isn't new ways of instantiating abstract classes and interfaces, it's a different mechanism to invoke functionality, as well as the ability to have one common signature for every interface method that's invoked.

> Using OOP, it isn't uncommon to have to create multiple classes just to create a simple "Hello World" plug-in.

Software That Almost Doesn't Exist

To see an example of just how far you can bring Active Events, check out Phosphorus Five at **github.com/polterguy/phosphorusfive**. Phosphorus Five is a large software project, but the Active Events implementation is only about 1.000 lines of code. You can find the Active Events implementation in core/p5.core.

In this project I created a new programming language, Hyperlambda, for creating Active Events hooks. This allows me to change the implementation of my for-each and while statements if I want. I can also scale out my else statements, to be executed on another server. And I can easily extend the programming language to create my own domain-specific keywords, for doing whatever my domain problems require.

I also created a file format that allows me to declare a Node structure dynamically, as a piece of text, and store it to my disk or my database. Furthermore, I created my own expression syntax, allowing me to reference any nodes within any part of my tree. This results in what I like to refer to as a non-programming language, yet one that's 100 percent Turing complete. My non-programming language is neither dynamically interpreted nor statically compiled and is sometimes able to do, with five lines of code, what requires hundreds of lines of code in other programming languages.

With Active Events, together with Hyperlambda and a managed AJAX library (containing just one AJAX widget), I've managed to create something that I don't even know how to talk about, without bringing in names such as "Web OS." There are roughly 30 projects in Phosphorus Five, and there are no references among the different plug-ins. All projects simply reference the p5.core, which is the Active Events implementation, as well as p5.exp, the expression engine. That's it. The main Web site project, p5.website in the core folder, contains only one simple container widget and almost no logic. All plug-ins are dynamically loaded in my Global.asax during application startup. Still, all projects dynamically invoke functionality inside of other projects. No references, no dependencies, no problems!

Back to Basics

The cure is always in the problem. Some of the very same problems OOP was invented to fix—global functions and data—paradoxically become the cure for the problems OOP unintentionally created. If you take a look at the Active Events design pattern, the first thing you realize is that, to some extent, this is going back to basics, but with global functions instead of methods and classes. However, because no signatures or types need to be known and shared between the consumer and the implementation of an active event, you have more of a black box environment than you might have with OOP. This allows you to easily exchange, for instance, SaveToDatabase with InvokeWebService or SaveToFile. No interfaces, no types, no POD structures, no classes, and only one commonly shared signature. Only good old, simple data. Data in, data out!

Polymorphism is as easy as changing a string. Here's an example of how to implement polymorphism using Active Events:

```
string myEvent = "some-active-event";
if (usePolymorphism) {
  myEvent = "some-other-active-event";
}
context.Raise (myEvent);
```

I realize this polymorphism construct must appear ridiculously naive and simple to the seasoned architect. However, its simplicity is the reason it works. With Active Events, you can fetch the name of a method or function from a database, a configuration file or even from a user providing its name through a textbox in your form. You can think of this as a variation of polymorphism that doesn't require explicit classes. This is polymorphism without types. This is polymorphism dynamically determined during runtime. By taking all traditional ideas about polymorphism out of the equation, and refactoring the very essence of polymorphism, you end up with actual working polymorphism—encapsulation and

polymorphism, without classes, types, interfaces or design patterns. Chaining your Active Events becomes as easy as combining atoms into molecules. *That* is agile software!

Node.cs, the Last Graph Type You'll Ever Need

With Active Events, you exclusively send data in and out from your active events. This is what lets you loosely couple your components. To do this, you need a data class—the only graph type you'll need when using an Active Events paradigm. Your class needs to be able to wrap all possible fields and properties from all possible classes. In Phosphorus Five, this class is called Node.cs and it's simply a graph object, with a key/value/child design.

The key to successfully implementing Active Events is that the Node class is the only argument your Active Events are allowed to take as input and return as output. It just so happens that almost all classes can be effectively reduced to a key/value/child graph POD object. This is what, combined with the dynamic loading of the Active Events assemblies, significantly reduces the number of dependencies between projects.

The Node.cs implementation needs to be able to hold a key or name, a value that can be any object, and a "children" collection of Nodes. If your Node class conforms to these constraints, you can easily transform almost all possible objects to a Node instance. For those acquainted with JSON or XML, the similarities may be obvious at this point. The following is simplified pseudo code demonstrating the Node class structure:

```
class Node
{
  public string Name;
  public object Value;
  public List<Node> Children;
}
```

Internally, within your components, you can use as much OOP as you wish. However, when a component needs to invoke logic in another component, all the input data must somehow be transformed to a Node instance. When information is returned from Active Events, the same needs to be done. But within your components, you're free to use any classes, interfaces, abstract factories, facade components, singletons, and design patterns you wish. Externally, though, there's only the Node, and Active Events becomes the bridge between the components. Think of Active Events as the protocol, and Nodes as its data, if it helps you to create a mental image of the relationship.

Wrapping Up

Although intended to replace most of the other design patterns in the world today, Active Events is not a silver bullet. Among other things, this technology comes with some overhead. For instance, instead of invoking methods directly, you go through a Dictionary lookup. In addition, the methods use some reflection. These indirect method invocations are probably orders of magnitudes more expensive than traditional virtual method invocation. Moreover, you need to transform your objects to nodes and data wherever you interface with other components.

However, active events are not supposed to replace everything you do. The idea is to provide better interfacing between components, and for this, the performance overhead is not a primary concern. Whether it takes 5 CPU cycles or 500 CPU cycles to invoke your SaveToDatabase method is completely irrelevant if its actual implementation takes 5,000,000 CPU cycles! Donald Knuth once said, "Premature optimization is the root of all evil."

Every time you think of writing an interface, you should ask yourself whether it would be better to create an active event instead. Once you've gotten some experience with Active Events, the answer to that question tends to become, "Probably yes." With Active Events, interfaces and abstract classes are for the most part reduced in emphasis.

I know that sounds like an absurdly bold statement, but check out the Phosphorus Five project discussed in "Software That Almost Doesn't Exist." In Hyperlambda, the "language" I created for Active Events, an object can be a file, a folder, a lambda callback, a subpart of a graph Node tree, a piece of text taken from a database, or a piece of data sent over HTTP. And all objects can be executed, as if they were machine-understandable execution trees. In Hyperlambda, you could, in theory, execute the number 42.

Although intended to replace most of the other design patterns in the world today, Active Events is not a silver bullet.

I first thought of Active Events more than seven years ago, and intuitively felt they held an intrinsic beauty. The problem is that they challenge 60 years of conventional programming wisdom. Active Events become the point where even our best practices must be refactored. It took me seven years to unlearn what I had previously learned and to trust my intuition. The biggest problem with Active Events is actually not in their implementation, it's in your head. The fact that I have entirely recreated Phosphorus Five five times is a testament to that fact.

In the first draft of this article, I created dozens of analogies. By hooking into your pre-existing knowledge about other subjects, such as physics and biology, I attempted to convince you about the superiority of Active Events. In my second draft, I tried to provoke you. The idea was to motivate you to prove me wrong, such that you would look for flaws in my statements, without being able to find any, of course. However, in my third and final draft, I decided to simply tell you about Active Events. If you can get your head to go along, Active Events will be the last design pattern you'll ever have to learn. ■

Thomas Hansen *has been creating software since he was 8 years old, when he started writing code using the Oric-1 computer in 1982. Occasionally he creates code that does more good than harm. His passions include the Web, AJAX, Agile methodologies and software architecture.*

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

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

7 LOCATIONS TO CHOOSE FROM

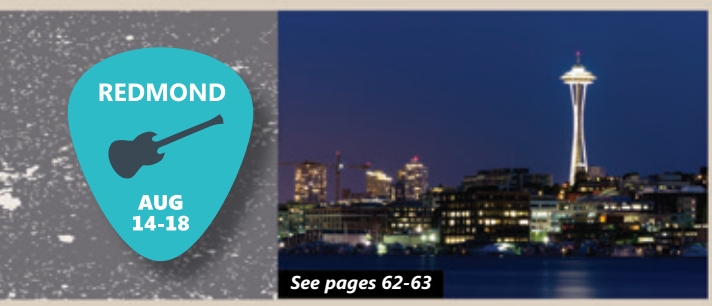
JOIN US

ROCK YOUR CODE TOUR 2017

See pages 68-71

See pages 36-37

See pages 38-39



See pages 62-63

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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

vslive.com

TURN THE PAGE FOR MORE EVENT DETAILS →

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

AUSTIN, TX
MAY 15-18, 2017
HYATT REGENCY

INTENSE TRAINING FOR DEVELOPERS, ENGINEERS, PROGRAMMERS, ARCHITECTS AND MORE!

HOT TOPICS INCLUDE:

- Visual Studio / .NET
- JavaScript / HTML5
- Angular
- Native Mobile & Xamarin
- Software Practices
- Database and Analytics
- ASP.NET Core
- Web API
- ALM / DevOps
- Cloud Computing
- UWP
- Unit Testing

FRIDAY, MAY 19: POST-CON HANDS-ON LABS

Choose From:

- Angular
- ASP.NET Core MVC/ Entity Framework

NEW! Only $595!

SPACE IS LIMITED

Register by March 17 and Save $300!*

Use promo code VSLMAR2 *Savings Based on 3-Day and 4-Day Packages Only.

REGISTER NOW

EVENT PARTNER Microsoft

GOLD SPONSOR Progress

SUPPORTED BY Visual Studio msdn magazine Visual Studio MAGAZINE

PRODUCED BY 1105 MEDIA YOUR GROWTH, OUR BUSINESS.

AUSTIN AGENDA AT-A-GLANCE

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

START TIME	END TIME	Visual Studio Live! Pre-Conference Workshops: Monday, May 15, 2017 *(Separate entry fee required)*		
9:00 AM	6:00 PM	**M01** Workshop: Distributed Cross-Platform Application Architecture - *Jason Bock & Rockford Lhotka*	**M02** Workshop: Practical ASP.NET DevOps with VSTS or TFS - *Brian Randell*	**M03** Workshop: Building for the Internet of Things: Hardware, Sensors & the Cloud - *Nick Landry*
6:45 PM	9:00 PM	Dine-A-Round		

START TIME	END TIME	Visual Studio Live! Day 1: Tuesday, May 16, 2017			
8:00 AM	9:00 AM	Keynote: To Be Announced - *Yochay Kiriaty, Principal Program Manager, Microsoft*			
9:15 AM	10:30 AM	**T01** Go Mobile With C#, Visual Studio, and Xamarin - *James Montemagno*	**T02** Angular 2 – The 75-Minute Crash Course - *Chris Klug*	**T03** What's New in Azure IaaS v2 - *Eric D. Boyd*	**T04** Tour of Visual Studio 2017 - *Jason Bock*
10:45 AM	12:00 PM	**T05** Building Connected and Disconnected Mobile Apps - *James Montemagno*	**T06** Enriching MVC Sites with Knockout JS - *Miguel Castro*	**T07** Bots are the New Apps: Building Bots with ASP.NET WebAPI & Language Understanding - *Nick Landry*	**T08** Getting to the Core of .NET Core - *Adam Tuliper*
12:00 PM	1:30 PM	Lunch - Visit Exhibitors			
1:30 PM	2:45 PM	**T09** Lessons Learned from Real World Xamarin.Forms Projects - *Nick Landry*	**T10** Explore Web Development with Microsoft ASP.NET Core 1.0 - *Mark Rosenberg*	**T11** To Be Announced	**T12** Tactical DevOps with VSTS - *Brian Randell*
3:00 PM	4:15 PM	**T13** Xamarin vs. Cordova - *Sahil Malik*	**T14** Assembling the Web - A Tour of WebAssembly - *Jason Bock*	**T15** Cloud Oriented Programming - *Vishwas Lele*	**T16** Exploring C# 7 New Features - *Adam Tuliper*
4:15 PM	5:30 PM	Welcome Reception			

START TIME	END TIME	Visual Studio Live! Day 2: Wednesday, May 17, 2017			
8:00 AM	9:15 AM	**W01** To Be Announced	**W02** Introduction to Writing Custom Angular (Not 2.0) Directives - *Miguel Castro*	**W03** Microservices with Azure Container Service & Service Fabric - *Vishwas Lele*	**W04** Database Continuous Integration - *Steve Jones*
9:30 AM	10:45 AM	**W05** Write Once, Run Everywhere - Cordova, Electron, and Angular2 - *Sahil Malik*	**W06** Integrating AngularJS & ASP.NET MVC - *Miguel Castro*	**W07** Go Serverless with Azure Functions - *Eric D. Boyd*	**W08** Continuous Integration and Deployement for Mobile Using Azure Services - *Kevin Ford*
11:00 AM	12:00 PM	General Session: To Be Announced - *Jeffrey T. Fritz, Senior Program Manager, Microsoft*			
12:00 PM	1:30 PM	Birds-of-a-Feather Lunch - Visit Exhibitors			
1:30 PM	2:45 PM	**W09** Building Cross-Platform Business Apps with CSLA. NET - *Rockford Lhotka*	**W10** Angular 2, ASP.NET Core and Gulp – Happily Forever After, or the Beginning of an Apocalypse? - *Chris Klug*	**W11** Using Cognitive Services in Business Applications - *Michael Washington*	**W12** .NET Deployment Strategies: The Good, The Bad, The Ugly - *Damian Brady*
3:00 PM	4:15 PM	**W13** Creating Great Looking Android Applications Using Material Design - *Kevin Ford*	**W14** Use Docker to Develop, Build, and Deploy Applications, A Primer - *Mark Rosenberg*	**W15** Busy Developer's Guide to the Clouds - *Ted Neward*	**W16** PowerShell for Developers - *Brian Randell*
4:30 PM	5:45 PM	**W17** SOLID – The Five Commandments of Good Software - *Chris Klug*	**W18** JavaScript for the C# (and Java) Developer - *Philip Japikse*	**W19** Introduction to Azure Machine Learning Studio (for the non-Data Scientist) - *Michael Washington*	**W20** Brownfields DevOps in Practice - *Damian Brady*
7:00 PM	9:00 PM	Rollin' On the River Bat Cruise			

START TIME	END TIME	Visual Studio Live! Day 3: Thursday, May 18, 2017			
8:00 AM	9:15 AM	**TH01** A Developers Introduction to HoloLens - *Billy Hollis*	**TH02** Extend and Customize the Visual Studio Environment - *Walt Ritscher*	**TH03** Entity Framework Core for Enterprise Applications - *Benjamin Day*	**TH04** Agile Failures: Stories From The Trenches - *Philip Japikse*
9:30 AM	10:45 AM	**TH05** Unity for .NET Developers - The Time is Now! - *John Alexander*	**TH06** SASS and CSS for Developers - *Robert Boedigheimer*	**TH07** A Tour of SQL Server 2016 Security Features - *Steve Jones*	**TH08** Distributed Architecture: Microservices and Messaging - *Rockford Lhotka*
11:00 AM	12:15 PM	**TH09** Take the Tests: Can You Evaluate Good and Bad Designs? - *Billy Hollis*	**TH10** Debugging Your Website with Fiddler and Chrome Developer Tools - *Robert Boedigheimer*	**TH11** Busy Developer's Guide to NoSQL - *Ted Neward*	**TH12** Top 10 Ways to Go from Good to Great Scrum Master - *Benjamin Day*
12:15 PM	1:30 PM	Lunch			
1:30 PM	2:45 PM	**TH13** Using Visual Studio to Scale your Enterprise - *Richard Hundhausen*	**TH14** Building Single Page Web Applications Using Aurelia.js and the MVVM Pattern - *Ben Hoelting*	**TH15** Azure Data Lake - *Michael Rys*	**TH16** Care and Feeding of Your Product Owner - *John Alexander*
3:00 PM	4:15 PM	**TH17** Windows Package Management with NuGet and Chocolatey - *Walt Ritscher*	**TH18** Tools for Modern Web Development - *Ben Hoelting*	**TH19** U-SQL Killer Scenarios: Performing Advanced Analytics and Big Cognition at Scale with U-SQL - *Michael Rys*	**TH20** Stop the Waste and Get Out of (Technical) Debt - *Richard Hundhausen*

START TIME	END TIME	Full Day Hands-On Labs: Friday, May 19, 2017 *(Separate entry fee required)*	
8:00 AM	5:00 PM	**HOL01** Full Day Hands-On Lab: Busy Developer's HOL on Angular - *Ted Neward*	**HOL02** Full Day Hands-On Lab: Develop an ASP.NET Core MVC/Entity Framework Core App in a Day - *Philip Japikse*

Speakers and sessions subject to change

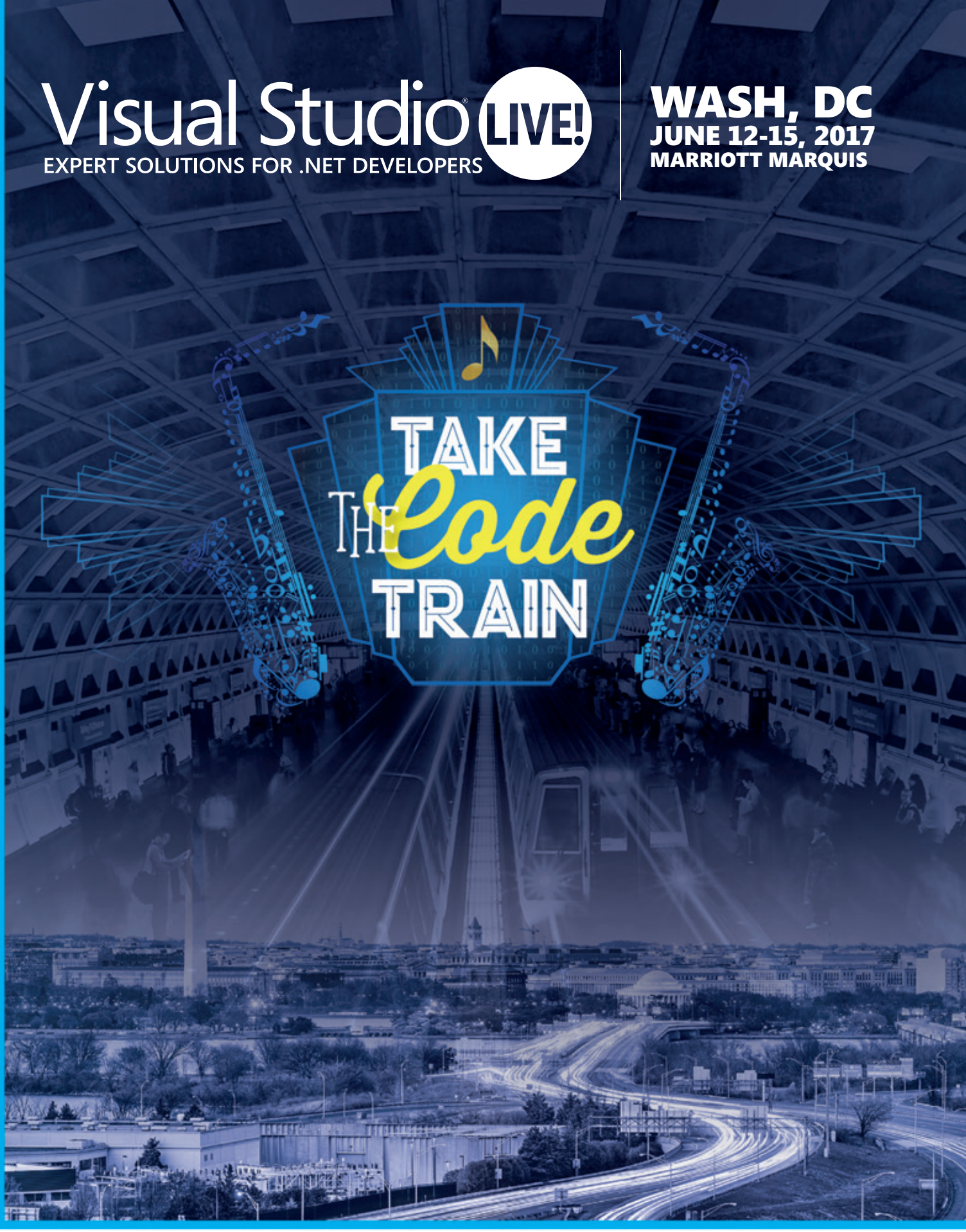
vslive.com/austinmsdn

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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



EVENT PARTNER

Microsoft

SUPPORTED BY

Visual Studio msdn magazine Visual Studio MAGAZINE

PRODUCED BY

1105 MEDIA
YOUR GROWTH. OUR BUSINESS.

INTENSE TRAINING FOR DEVELOPERS, ENGINEERS, PROGRAMMERS, ARCHITECTS AND MORE!

Development Topics include:

- Visual Studio / .NET
- JavaScript / HTML5
- Angular
- Native Mobile & Xamarin
- Software Practices
- Database and Analytics
- ASP.NET Core
- Web API
- ALM / DevOps
- Cloud Computing
- UWP
- Unit Testing

Register by April 21 and Save $300!

Use promo code VSLMAR2

REGISTER NOW

ROCK YOUR CODE TOUR 2017

WASH. DC

JUNE 12-15

"I like that this event is in D.C. so I can actually attend! I learned about features that we could leverage as well as information about trends we are seeing in the industry at the cutting edge." – Jonathan Cogan, Geico

"Very intelligent speakers. I really liked the topics covering new .NET and C# tools/technology and code." – Jon Bleichner, NIH

vslive.com/dcmsdn

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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

Simplify Safe Array Programming in C++ with CComSafeArray

Giovanni Dicanio

It's common to develop complex software systems building components written in different languages. For example, you might have some high-performance C++ code embedded in a C-interface, dynamic-link library (DLL) or in a COM component. Or, you can write a Windows service in C++ that exposes some COM interfaces. Clients written in C# can interact with that service via COM interop.

Often, you want to exchange some data in the form of arrays between those components. For example, you might have some C++ components that interact with some hardware and produce an array of data, like an array of bytes representing the pixels of an image read from an input device, or an array of floating point numbers representing measurements read from a sensor. Or, you might have a Windows service written in C++ that interacts with other low-level modules and returns arrays of strings that you want to consume in GUI clients written in C#, or in a scripting language. Passing that data across module boundaries isn't trivial and requires the use of well-designed and well-crafted data structures.

The Windows programming platform offers a convenient ready-to-use data structure that can be used for that purpose: the SAFEARRAY, whose definition can be found on the Windows Dev Center (bit.ly/2fLXY6K). Basically, the SAFEARRAY data structure describes a particular instance of a safe array, specifying attributes such as its number of dimensions and a pointer to the actual safe array's data. A safe array is usually handled in code via a pointer to its SAFEARRAY descriptor, that is, SAFEARRAY*. There are also C-interface Windows APIs for manipulating safe arrays, such as SafeArrayCreate and SafeArrayDestroy for creation and destruction, and other functions to lock a safe array instance and safely access its data. For further details on the SAFEARRAY C data structure and some of its native C-interface APIs, see the online companion piece to this article, "Introducing the SAFEARRAY Data Structure" (msdn.com/magazine/mt778923).

However, for C++ programmers, instead of working at the C-interface level, it's more convenient to use higher-level C++ classes, such as the Active Template Library (ATL) CComSafeArray.

In this article, I'll discuss C++ code samples of increasing complexity to create safe arrays storing different kinds of data using ATL helper classes, including CComSafeArray.

This article discusses:

- Wrapping the raw C SAFEARRAY data structure using the higher-level ATL CComSafeArray C++ class template
- Producing safe arrays of scalar types such as bytes
- Producing safe arrays of BSTR strings
- Producing safe arrays of variants
- Some useful C# PInvoke declarations for safe arrays

Technologies discussed:

C++, Win32, Safe Arrays, ATL, CComSafeArray, BSTR, VARIANT, PInvoke

Safe Array vs. STL Vector

Standard Template Library (STL) class templates such as std::vector are excellent containers for C++ code *inside* module boundaries, and I encourage you to use those in such contexts. For example, it's more efficient to dynamically add content and grow a std::vector than a safe array, and std::vector is nicely integrated with algorithms from the STL and other third-party cross-platform C++ libraries (such as Boost), as well. Moreover, using std::vector lets you develop cross-platform standard C++ code; instead, safe arrays are Windows-platform specific.

On the other hand, when safe arrays win, it's at the module boundaries. In fact, std::vectors can't safely cross module boundaries and can't be consumed by clients written in languages different than C++. Instead, these are the very contexts in which it makes sense to use safe arrays. A good coding pattern is one in which you perform the core processing using standard C++ and STL containers such as std::vector. Then, when you need to transfer such array data across module boundaries, you can project the content of std::vector to a safe array that's an excellent candidate for crossing module boundaries and for being consumed by clients written in languages different than C++.

When a CComSafeArray object goes out of scope, its destructor will *automatically* clean up the memory and resources allocated by the wrapped safe array.

The ATL CComSafeArray Wrapper

The safe array "native" programming interface uses Win32 C-interface APIs, as described in the online companion piece to this article. While it's possible to use those C functions in C++ code, they tend to lead to cumbersome and bug-prone code; for example, you have to pay attention to properly match safe array deallocations with every safe array allocation to properly match locks with unlocks, and so on.

Thankfully, with C++ it's possible to simplify that C-style code quite a bit, using convenient coding patterns like RAII and destructors. For example, you can write a class to wrap raw SAFEARRAY descriptors, and have the constructor call SafeArrayLock, and the destructor call the matching SafeArrayUnlock function. In this way, the safe array is automatically unlocked once the wrapper object goes out of scope. It makes sense to shape this class as a template, as well, to enforce type safety on the safe array. In fact, while in C the safe array's genericity is expressed using a void pointer for the SAFEARRAY::pvData field, in C++ you can do better type checking at compile time using templates.

Fortunately, you don't have to write such a class template from scratch. In fact, ATL already provides a convenient C++ class template to simplify safe array programming: CComSafeArray, declared in the <atlsafe.h> header. In the ATL CComSafeArray<T>, the T template parameter represents the type of data stored in the safe array. For example, for a safe array of BYTEs you would use CComSafeArray<BYTE>; a safe array of floats is wrapped using CComSafeArray<float> and so on. Note that the internal-wrapped safe array is still a polymorphic void-pointer-based C-style array. However, the C++ wrapping layer built by CComSafeArray offers a higher and safer level of abstraction that includes both safe array automatic lifetime management and better type safety than C void*.

CComSafeArray<T> has just one data member, m_psa, which is a pointer to a safe array descriptor (SAFEARRAY*) wrapped by the CComSafeArray object.

Constructing CComSafeArray Instances

The CComSafeArray default constructor creates a wrapper object that contains just a null pointer to a safe array descriptor; basically, it wraps nothing.

In addition, there are several constructor overloads that can come in handy. For example, you can pass an element count to create a safe array made by a given number of elements:

```
// Create a SAFEARRAY containing 1KB of data
CComSafeArray<BYTE> data(1024);
```

It's also possible to specify a lower bound different than zero, which is the default:

```
// Create a SAFEARRAY containing 1KB of data
// with index starting from 1 instead of 0
CComSafeArray<BYTE> data(1024, 1);
```

However, when you write code to operate on safe arrays that are meant to be processed by C++ or C#/.NET clients, it's better to stick with the usual convention of having a lower bound of zero (instead of one).

Note that CComSafeArray constructors automatically call SafeArrayLock for you; so, the wrapped safe array is already locked and ready for read-and-write operations in user code.

If you have a pointer to an existing safe array descriptor, you can pass it to another CComSafeArray constructor overload:

```
// psa is a pointer to an existing safe array descriptor
// (SAFEARRAY* psa)
CComSafeArray<BYTE> sa(psa);
```

In this case, CComSafeArray attempts to create a deep copy of the original safe array of BYTEs pointed by "psa."

Automatic Resource Cleanup with CComSafeArray

When a CComSafeArray object goes out of scope, its destructor will *automatically* clean up the memory and resources allocated by the wrapped safe array. This is very convenient and simplifies the C++ code a lot. In this way, C++ programmers can focus on the core of their code instead of the details of safe array memory clean up, avoiding nasty memory leak bugs.

Spelunking inside the ATL CComSafeArray code, you'll see that the CComSafeArray destructor calls SafeArrayUnlock first, and then SafeArrayDestroy, so there's no need for client code to explicitly unlock and destroy the safe array; actually, that would be a double-destruction bug.

Convenient Methods for Common SafeArray Operations

In addition to automatic safe array lifetime management, the CComSafeArray class template offers some convenient methods

to simplify operations on safe arrays. For example, you can simply call the GetCount method to get the element count of the wrapped safe array. To get access to the safe array's items, you can call the CComSafeArray::GetAt and SetAt methods, simply specifying the item's index.

For example, to iterate through the elements in an existing safe array, you can use code like this:

```
// Assume sa is a CComSafeArray instance wrapping an existing safe array.
// Note that this code is generic enough to handle safe arrays
// having lower bounds different than the usual zero.
const LONG lb = sa.GetLowerBound();
const LONG ub = sa.GetUpperBound();

// Note that the upper bound is *included* (<=)
for (LONG i = lb; i <= ub; i++)
{
  // ... use sa.GetAt(i) to access the i-th item
}
```

In addition, CComSafeArray overloads operator[] to offer even a simpler syntax for accessing safe array's items. You'll see some of those methods in action in the next sections of this article.

It's also possible to append new items to an existing safe array, invoking the CComSafeArray::Add method. Moreover, CComSafeArray::Resize, as its name clearly suggests, can be used to resize the wrapped safe array. However, in general, I'd suggest you write C++ code that operates on std::vector, which is faster than safe arrays at resizing and is well integrated with other STL algorithms and other third-party cross-platform C++ libraries, as well. Then, once the operations on the content of std::vector are complete, the data can be copied to a safe array, which can safely cross module boundaries (unlike std::vector), and can be consumed also by clients written in languages different than C++.

Safe array copy operations are possible using the overloaded copy-assignment operator= or invoking methods such as CComSafeArray::CopyFrom.

> Safe arrays can safely cross module boundaries (unlike std::vector), and can be consumed also by clients written in languages different than C++.

'Move Semantics' in CComSafeArray

The CComSafeArray class doesn't implement "move semantics" in the strict C++11 sense; in fact, this ATL class predates C++11, and at least up until Visual Studio 2015, C++11 move operations such as move constructor and move assignment operator haven't been added to it. However, there's a different kind of move semantics offered by CComSafeArray and it's based on a couple of methods: Attach and Detach. Basically, if you have a pointer to a safe array descriptor (SAFEARRAY*), you can pass it to the Attach method, and the CComSafeArray will take *ownership* of that raw safe array. Note that any previously existing safe array data wrapped in the CComSafeArray object is properly cleaned up.

Similarly, you can call the CComSafeArray::Detach method and the CComSafeArray wrapper will release the ownership of the wrapped safe array to the caller, returning a pointer to the previously owned safe array descriptor. The Detach method comes in handy when you create a safe array in C++ code using CComSafeArray, and then you hand that as an output pointer parameter to a caller, for example in a COM interface method or in a C-interface DLL function. In fact, the CComSafeArray C++ class cannot cross COM or C-interface DLL boundaries; only raw SAFEARRAY* descriptor pointers can.

C++ Exceptions Cannot Cross COM and C DLL Boundaries

Some of CComSafeArray methods such as Create, CopyFrom, SetAt, Add, and Resize return HRESULTs to signal success or error conditions, as is customary in COM programming. However, other methods such as some CComSafeArray constructor overloads, or GetAt, or operator[], actually throw exceptions on errors. By default, ATL throws C++ exceptions of type CAtlException, which is a tiny wrapper around an HRESULT.

However, C++ exceptions cannot cross COM method or C-interface DLL function boundaries. For COM methods, it's mandatory to return HRESULTs to signal errors. This option can be used in C-interface DLLs, as well. So, when writing C++ code that uses the CComSafeArray wrapper (or any other throwing component, as well), it's important to guard that code using a try/catch block. For example, inside the implementation of a COM method, or inside a DLL boundary function that returns an HRESULT, you can write code like this:

```
try
{
  // Do something that can potentially throw exceptions as CAtlException ...
}
catch (const CAtlException& e)
{
  // Exception object implicitly converted to HRESULT,
  // and returned as an error code to the caller
  return e;
}

// All right
return S_OK;
```

Producing a Safe Array of Bytes

After the previous conceptual framework introducing safe arrays and the convenient C++ CComSafeArray ATL wrapper, I'd like to discuss some practical applications of safe array programming, showing CComSafeArray in action. I'll start with a simple case: producing a safe array of bytes from C++ code. This safe array can be passed as an output parameter in a COM interface method or C-interface DLL function.

A safe array is typically referenced via a pointer to its descriptor, which translates to SAFEARRAY* in C++ code. If you have an *output* safe array parameter, you need another level of indirection, so a safe array passed as output parameter has the SAFEARRAY** form (double pointer to SAFEARRARY descriptor):

```
STDMETHODIMP CMyComComponent::DoSomething(
    /* [out] */ SAFEARRAY** ppsa)
{
  // ...
}
```

GROUPDOCS
Document Manipulation APIs

Manipulating Files?

APIs to view, export, annotate, compare, sign, automate and search documents in your applications.

GroupDocs.Total

GroupDocs.Viewer
GroupDocs.Annotation
GroupDocs.Conversion
GroupDocs.Comparison
GroupDocs.Signature
GroupDocs.Assembly
GroupDocs.Metadata
GroupDocs.Search
GroupDocs.Text

Try for Free

.NET Libraries

Java Libraries

Cloud APIs

Contact Us:
US: +1 903 306 1676
EU: +44 141 628 8900
AU: +61 2 8006 6987
sales@asposeptyltd.com

Visit us at www.groupdocs.com

GROUPDOCS
Document Manipulation APIs

Inside the COM method, don't forget to use a try/catch guard to catch exceptions and translate them to corresponding HRESULTs, as shown in the previous paragraph. (Note that the STDMETHODIMP preprocessor macro implies that the method returns an HRESULT.)

Inside the try block, an instance of the CComSafeArray class template is created, specifying BYTE as the template parameter:

```
// Create a safe array storing 'count' BYTEs
CComSafeArray<BYTE> sa(count);
```

Then you can simply access the elements in the safe array using CComSafeArray's overloaded operator[], for example:

```
for (LONG i = 0; i < count; i++)
{
  sa[i] = /* some value */;
}
```

Once you've completely written your data to the safe array (for example, copying it from a std::vector), the safe array can be returned as an output parameter, invoking the CComSafeArray Detach method:

```
*ppsa = sa.Detach();
```

Note that after invoking Detach, the CComSafeArray wrapper object transfers the safe array ownership to the caller. So, when the CComSafeArray object goes out of scope, the safe array created in the previous code is *not* destroyed. Thanks to Detach, the safe array data was just transferred, or "moved" from the previous code that created the safe array in the first place, to the caller. It's now the caller's responsibility to destroy the safe array when it's no longer needed.

Safe arrays are convenient to exchange array data across DLL module boundaries, as well. Think, for example, of a C-interface DLL written in C++ that produces a safe array that's consumed by some C# client code. If the boundary function exported by the DLL has this prototype:

```
extern "C" HRESULT __stdcall ProduceByteArrayData(/* [out] */ SAFEARRAY** ppsa)
```

the corresponding C# PInvoke declaration is:

```
[DllImport("NativeDll.dll", PreserveSig = false)]
public static extern void ProduceByteArrayData(
  [Out, MarshalAs(UnmanagedType.SafeArray, SafeArraySubType = VarEnum.VT_UI1)]
  out byte[] result);
```

UnmanagedType.SafeArray means that the actual native array type at the function boundary is a SAFEARRAY. The VarEnum.VT_UI1 value assigned to SafeArraySubType specifies that the type of the data stored in the safe array is BYTE (which is an unsigned integer of size exactly of one byte). For a safe array storing 4-byte-signed integers, on the C++ side you'd have CComSafeArray<int>, and the corresponding PInvoke VarEnum type would be VT_I4 (meaning signed integer of 4-byte size). The safe array is mapped to a byte[] array in C#, and that's passed as an out parameter.

The PreserveSig = false attribute tells PInvoke to translate the error HRESULTs returned by the native function to exceptions in C#.

Figure 1 shows a complete sample code snippet for producing a safe array of bytes in C++ using CComSafeArray. The code is part of a hypothetical COM method; however, the same CComSafeArray-based code can be used for C-interface DLL boundary functions, as well.

Producing a Safe Array of Strings

So now you've learned how to create a safe array of bytes and even the PInvoke declaration signature to use in C# when the safe array is passed as an output parameter in a C-interface DLL function. This coding pattern works well for safe arrays storing other scalar types such as ints, floats, doubles and so on. Those types have the same binary representations in both C++ and C#, and are easily marshalled between these two worlds and across COM component boundaries, as well.

However, some additional attention should be paid to safe arrays that store strings. Strings require special care, as they're more complex than just single scalars such as bytes or integers or floating-point numbers. The BSTR type is a convenient one used to represent strings that can safely cross module boundaries. While this type is quite versatile, you can think of it as a length-prefixed Unicode UTF-16 string pointer. The default Marshaller knows how to copy BSTRs and how to make them cross COM or C-interface function boundaries. An interesting and detailed description on BSTR semantics can be read in an Eric Lippert blog post at bit.ly/2fLXTfY.

To create a safe array storing strings in C++, the CComSafeArray class template can be instantiated using the BSTR type:

```
// Creates a SAFEARRAY containing 'count' BSTR strings
CComSafeArray<BSTR> sa(count);
```

While the safe array type specified here is the raw BSTR C type, in C++ code it's much better (that is, easier and safer) to use an RAII wrapper around raw BSTRs. ATL offers such a convenient wrapper in the form of the CComBSTR class. In Win32/C++ code compiled with Microsoft Visual C++ (MSVC), Unicode UTF-16 strings can be represented using the std::wstring class. So, it makes sense to build a convenient helper function to convert from a std::wstring to ATL::CComBSTR:

```
// Convert from STL wstring to the ATL BSTR wrapper
inline CComBSTR ToBstr(const std::wstring& s)
{
  // Special case of empty string
  if (s.empty())
  {
    return CComBSTR();
  }

  return CComBSTR(static_cast<int>(s.size()), s.data());
}
```

Figure 1 **Producing a Safe Array of Bytes Using CComSafeArray**

```
STDMETHODIMP CMyComComponent::DoSomething(/* [out] */ SAFEARRAY** ppsa) noexcept
{
  try
  {
    // Create a safe array storing 'count' BYTEs
    const LONG count = /* some count value */;
    CComSafeArray<BYTE> sa(count);

    // Fill the safe array with some data
    for (LONG i = 0; i < count; i++)
    {
      sa[i] = /* some value */;
    }

    // Return ("move") the safe array to the caller
    // as an output parameter
    *ppsa = sa.Detach();
  }
  catch (const CAtlException& e)
  {
    // Convert ATL exceptions to HRESULTs
    return e;
  }

  // All right
  return S_OK;
}
```

STAY CONNECTED.

msdn magazine

MSDN.MICROSOFT.COM

To fill a CComSafeArray<BSTR> with strings copied from an STL vector<wstring>, the vector can be iterated through, and for each wstring in the vector, a corresponding CComBSTR object can be created invoking the aforementioned helper function:

```
// 'v' is a std::vector<std::wstring>
for (LONG i = 0; i < count; i++)
{
  // Copy the i-th wstring to a BSTR string
  // safely wrapped in ATL::CComBSTR
  CComBSTR bstr = ToBstr(v[i]);
```

Then, the returned bstr can be copied in the safe array object, invoking the CComSafeArray::SetAt method:

```
  hr = sa.SetAt(i, bstr);
```

The SetAt method returns an HRESULT, so it's a good programming practice to check its value and throw an exception in case of errors:

```
  if (FAILED(hr))
  {
    AtlThrow(hr);
  }

} // For loop
```

The exception will be converted to an HRESULT at the COM method or C-interface DLL function boundary. As an alternative, the error HRESULT could be returned directly from the previous code snippet.

The main difference of this BSTR safe array case with respect to the previous CComSafeArray<BYTE> sample is the creation of an intermediate CComBSTR wrapper object around the BSTR strings. There's no need for such wrappers for simple scalar types such as bytes, integers or floating-point numbers; but for more complex types such as BSTRs that require proper lifetime management, those C++ RAII wrappers come in handy. For example, wrappers such as CComBSTR hide calls to functions such as SysAllocString, which is used to create a new BSTR from an existing C-style string pointer. Similarly, the CComBSTR destructor automatically calls SysFreeString to release the BSTR memory. All these BSTR lifetime management details are conveniently hidden inside the CComBSTR class implementation, so C++ programmers can focus their attention on the higher-level logic of their code.

'Move Semantics' Optimization for Safe Arrays of BSTRs

Note that the previous CComSafeArray<BSTR>::SetAt method call performs a deep copy of the input BSTR into the safe array. In fact, the SetAt method has an additional BOOL bCopy third parameter, which is defaulted to TRUE. This bCopy flag parameter isn't important for scalar types such as bytes or integers of floating-point numbers, as they're all deep copied in the safe array. However, it's important for more complex types, such as BSTRs that have a non-trivial lifetime management and non-trivial copy semantics. For example, in this case of CComSafeArray<BSTR>, if FALSE is specified as third parameter to SetAt, the safe array simply takes ownership of the input BSTR, instead of deep copying it. It's kind of a snappy optimized move operation, instead of a deep copy. This optimization also requires that the Detach method is invoked on the CComBSTR wrapper to properly transfer the BSTR ownership from CComBSTR to the CComSafeArray:

```
hr = sa.SetAt(i, bstr.Detach(), FALSE);
```

As already shown for the CComSafeArray<BYTE> sample, once the creation of the CComSafeArray<BSTR> is completed, the safe array can be passed (moved) to the caller with code like this:

```
// Return ("move") the safe array to the caller
// as an output parameter (SAFEARRAY **ppsa)
*ppsa = sa.Detach();
```

A C-interface DLL function that builds a safe array of BSTR strings and passes that to the caller can be PInvoked in C# like this:

```
[DllImport("NativeDll.dll", PreserveSig = false)]
public static extern void BuildStringArray(
  [Out, MarshalAs(UnmanagedType.SafeArray, SafeArraySubType = VarEnum.VT_BSTR)]
  out string[] result);
```

Note the use of VarEnum.VT_BSTR to indicate the presence of a safe array storing BSTR strings. The SAFEARRAY of BSTRs produced in native C++ code is marshalled to C# using a string[] array type, passed as an out parameter.

Producing a Safe Array of Variants That Contain Strings

Let's further increase the level of complexity by another step. In addition to safe array storing elements of types such as byte, integer and BSTR strings, it's also possible to create a safe array of a "generic" type—the VARIANT type. A variant is a polymorphic type that can store values of a wide variety of different types, ranging from integers, to floating-point numbers, to BSTR strings and so on. The VARIANT C type is basically a gigantic union; its definition can be found at bit.ly/2fMc4Bu. Just like for the BSTR C type, ATL offers a convenient C++ wrapper around the raw C VARIANT type: the ATL::CComVariant class. In C++ code, it's simpler and safer to handle variants using the CComVariant wrapper, instead of directly invoking C functions for allocating, copying and clearing variants.

While clients written in C++ and C# understand safe arrays storing "direct" types (like shown in previous examples of BYTE and BSTR), there are some scripting clients that only understand safe array storing variants. So, if you want to build array data in C++ and you want to make that data consumable by such scripting clients, this data must be packed in a safe array storing variants, adding a new level of indirection (and some additional overhead, as well). Each variant item in the safe array will in turn store either a BYTE, a floating-point number, a BSTR or a value of whatever supported type.

Suppose that you want to modify the previous code that builds a safe array of BSTR strings, using a safe array of variants instead. The variants will in turn contain BSTRs, but what will be produced is a safe array of *variants*, not a direct safe array of BSTR strings.

First, to create a safe array of variants, the CComSafeArray class template can be instantiated this way:

```
// Create a safe array storing VARIANTs
CComSafeArray<VARIANT> sa(count);
```

Then, you can iterate through a set of strings stored, for example, in an STL vector<wstring>, and for each wstring you can create a CComBSTR object, just like in the previous code sample :

```
// 'v' is a std::vector<std::wstring>
for (LONG i = 0; i < count; i++)
{
  // Copy the i-th wstring to a BSTR string
  // safely wrapped in ATL::CComBSTR
  CComBSTR bstr = ToBstr(v[i]);
```

Now there comes a difference from the previous case of safe arrays of BSTRs. In fact, because this time you're building a safe

array of VARIANTs (not a direct safe array of BSTRs), you can't store the BSTR object directly in the CComSafeArray calling SetAt. Instead, first a variant object must be created to wrap that bstr; and then that variant object can be inserted into the safe array of variants:

```
    // First create a variant from the CComBSTR
    CComVariant var(bstr);

    // Then add the variant to the safe array
    hr = sa.SetAt(i, var);
    if (FAILED(hr))
    {
      AtlThrow(hr);
    }
  } // For loop
```

Note that CComVariant has an overloaded constructor taking a const wchar_t* pointer, so it would be possible to directly build a CComVariant from a std::wstring, invoking the wstring c_str method. However, in this case the variant will store only the initial chunk of the original wstring up until the first NUL-terminator; while both wstring and BSTR can potentially store strings with embedded NULs. Therefore, building an intermediate CComBSTR from the std::wstring, as previously done in the custom ToBstr helper function, covers this more generic case, as well.

As usual, to return the created safe array to the caller as a SAFEARRAY** output parameter, the CComSafeArray::Detach method can be used:

```
// Transfer ownership of the created safe array to the caller
*ppsa = sa.Detach();
```

If the safe array is passed via a C-interface function like this:

```
extern "C" HRESULT __stdcall BuildVariantStringArray(/* [out] */ SAFEARRAY** ppsa)
```

the following C# PInvoke declaration can be used:

```
[DllImport("NativeDll.dll", PreserveSig = false)]
pubic static extern void BuildVariantStringArray(
  [Out, MarshalAs(UnmanagedType.SafeArray, SafeArraySubType = VarEnum.VT_VARIANT)]
  out string[] result);
```

Note the use of VarEnum.VT_VARIANT for the SafeArraySubType, as this time the safe array created in C++ contains variants (that in turn wrap BSTR strings), but not BSTR directly.

In general, I'd suggest you export data using safe arrays of direct types instead of safe arrays storing variants, unless you're constrained by making your data accessible by scripting clients that can only handle the safe array of variants.

Wrapping Up

The safe array data structure is a convenient tool to exchange array data across different module and language boundaries. The safe array is a versatile data structure, and it's possible to store in it simple primitive types such as bytes, integers or floating-point numbers, but also more complex types such as BSTR strings, or even generic VARIANTs. This article showed how to simplify the programming of such data structures in C++ with concrete sample cases using ATL helper classes. ■

Giovanni Dicanio *is a computer programmer specializing in C++ and the Windows OS, a Pluralsight author (bit.ly/GioDPS) and a Visual C++ MVP. Besides programming and course authoring, he enjoys helping others on forums and communities devoted to C++. He can be contacted at giovanni.dicanio@gmail.com. He also blogs on msmvps.com/gdicanio.*

Thanks to the following technical experts for reviewing this article: *David Cravey and Marc Gregoire*

Immutable Collections

Hadi Brais

Side effects compromise the understandability and correctness of code. A method that mutates global or static variables has side effects. A method that mutates some of its parameters has side effects. If you want to understand a piece of code, you have to go through the code of all methods that are called and have side effects. Methods with side effects require thread synchronization to execute correctly when there are multiple threads.

What if you write methods that don't have side effects? What would the code look like and how would it perform? You can find out by making instances immutable so that side effects can't occur.

Generally, when an instance of a type is described to be immutable, it means that its value never changes. Immutability, like many things in software engineering, is a design choice. You don't really have to use it, but in some scenarios, you can benefit from it in terms of code performance or understandability. It's actually often useful that it's one of the fundamental principles of the successful functional programming paradigm. Considering that F# is a functional-first language, all instances are immutable unless explicitly specified otherwise. On the other hand, C# is an object-oriented-first language in which all instances are mutable unless explicitly specified otherwise. In this article, I'll show you how to leverage immutability in C#. But first, I'll define immutability in the context of this article.

This article discusses:

- Immutability and immutable types
- The immutable collections .NET library
- Comparison between mutable, immutable and concurrent collections

Technologies discussed:

C#, Immutable Collections Library, Visual Studio Diagnostic Tools, PerfView

Defining Immutability

Technically, there are many kinds of immutability. Any type that somewhat restricts changes to its state or the state of its instances can be described as immutable in some sense. The System.String type is an immutable type in the sense that the size of the string, the characters, and their order can't change. The System.MulticastDelegate type, which is the parent of all delegate types, is immutable just like System.String. Both use an array as an underlying data structure and make a copy of it to satisfy a requested change, no matter how small the change is. For more information on the kinds of immutability, please refer to the article at bit.ly/2kGVx4Z.

The System.Collections.ObjectModel.ReadOnlyCollection<T> is not immutable but it implements an immutable interface for a given mutable IList<T> object. This interface doesn't let the consumer change the number of items in the collection or their relative order. However, it doesn't say anything about immutability of the individual items, which depends on the whole type

hierarchy of T. Of course, code that has a reference to the underlying list can change it without constraints.

The immutable collections discussed in this article provide yet a different kind of immutability. To motivate the need for them, consider the following example:

A typical text editor provides several features or tools (such as spellcheck or code analysis) that analyze or process text that the user has written. With the prevalence of multicore computers, these tools can be run in the background while the user types. If you're not careful, you might run into thread-safety issues. The background analyzer reads the buffer containing the text at the same time the user is modifying it. Now, imagine instead of a buffer, the background process would logically get a snapshot of the text.

How can you achieve this correctly and efficiently? Using a type like String correctly solves the problem, but not efficiently. This would let the user change the text concurrently while the tool is running, but with every change, a new copy of the text is made, which can be slow and wasteful of memory for large documents. Another correct solution would be to use a mutable type, such as System.Text.StringBuilder. But this is also inefficient because the tool needs to make a copy of it under an acquired lock and the user wouldn't be able to make any changes until a copy is made. Using ReadOnlyCollection<T> wouldn't do any good, either, because the underlying collection is mutable and shared.

You need a different kind of immutability that enables you to make changes safely without using expensive thread synchronization mechanisms, while sharing as much data as possible between threads with no or little copying required. Immutable collections provide exactly this kind of immutability, known as persistence. They're not only useful in the scenario described earlier, but also in other multi- and single-threaded scenarios, as you'll see later in this article.

This article provides a detailed discussion of the design, implementation and performance of immutable collections to enable you to effectively use them and even write your own immutable collections and types. These collections can be used on any .NET platform that supports the .NET Standard 1.0 and later (which means you can use them on all Windows platforms, Xamarin and .NET Core). Immutable collections are relatively new and distributed as a NuGet package, so they aren't used by the .NET Framework itself, even though there are many framework APIs for which immutable collections would've been beneficial. Instead, such APIs use the potentially less-ideal ReadOnlyCollection<T> or use a copy of a mutable collection. I'll use the package version 1.3.0. The source code is available as part of CoreFX.

Note that it's possible to use unsafe code or Reflection to break almost any immutability guarantees. In general, when a type is described to be immutable, there's an implicit caveat made that these techniques can circumvent any immutability guarantees. This applies to the immutable collections discussed in this article.

Defining Immutable Collections

Before I discuss the internals of immutable collections, I have to define what they are. An immutable collection is a collection of instances that preserves its structure all the time and disallows element-level assignments, while still offering APIs to perform mutations. By the structure of a collection, I mean the number of elements and their relative order (which is determined by their indices in the case of an array structure and by their links in the case of a linked structure).

For example, if you push an element onto an ImmutableStack<T>, you'll get two isolated immutable stacks: one with the new element and one without. On the other hand, pushing an element onto a mutable Stack<T> effectively changes the stack and you'll only have one stack that contains the new element. Note that both immutable and mutable collections do not offer any guarantees regarding the elements themselves. If T was Int32 or String, then the elements would be immutable, as well. But if T was something like StringBuilder, then the elements are very much mutable.

In order to construct any immutable object, it has to be initialized. Therefore, during initialization, the object is mutable. Once a reference to the object has been published (by returning it from a non-private method), the object becomes effectively immutable for the rest of its lifetime.

The immutable collections are designed with two goals in mind. First, to reuse as much memory as possible, to avoid copying, and to reduce pressure on the garbage collector (such implementation is usually called persistent). The second goal is to support the same operations offered by mutable collections with competitive time complexities.

Immutable Stacks

The mutable Stack<T> type is implemented using an array. Arrays, however, aren't suitable for immutable collections because the only way to preserve the current instance is by copying the whole array and performing the change on that new array. This would make the immutable stack unacceptably slow. Linked lists can be elegantly used to implement it. Each element contains a pointer to the element below it (or null for the bottom element). An immutable stack is represented as a pointer to the top element. This way, you can push and pop elements of a given stack without changing it, while at the same time sharing all of its elements with the resulting stack. This simple design makes the immutable stack the simplest immutable collection. I show the differences between the mutable and immutable stacks further in this article.

Let's see how to create and use immutable stacks. The ImmutableStack<T> and all other immutable collections are defined in the System.Collections.Immutable namespace. To maximize sharing of memory, immutable collections don't offer any public constructors. In order to create an instance of an immutable collection, you have to use one of the CreateXxx<T> methods defined in a static type that corresponds to the immutable collection. For the immutable stack, this type is called ImmutableStack and it offers the following factory methods:

```
public static ImmutableStack<T> Create<T>();
public static ImmutableStack<T> Create<T>(T item);
public static ImmutableStack<T> Create<T>(params T[] items);
public static ImmutableStack<T> CreateRange<T>(IEnumerable<T> items);
```

All the methods have a generic type parameter T that specifies the type of the items stored in the collection. The first method creates an empty immutable stack, which internally simply returns the

singleton ImmutableStack<T>.Empty. The second method creates a stack with the specified item pushed onto it, which is equivalent to ImmutableStack<T>.Empty.Push(item). The third and fourth methods create a stack with the specified items pushed onto it in order. The CreateRange<T> method is implemented as follows:

```
var stack = ImmutableStack<T>.Empty;
foreach (var item in items)
{
  stack = stack.Push(item);
}

return stack;
```

All the factory methods for all the immutable collections are provided for convenience. All of them internally start with the empty collection and add the specified items to it. The items are always copied shallowly.

Now consider the following sequence of operations beginning with the empty stack:

```
ImmutableStack<Int32> s1 = ImmutableStack<Int32>.Empty;
ImmutableStack<Int32> s2 = s1.Push(1);
ImmutableStack<Int32> s3 = s2.Push(2);
ImmutableStack<Int32> s4 = s3.Push(3);
ImmutableStack<Int32> s5 = s4.Push(4);
ImmutableStack<Int32> s6 = s4.Pop();
ImmutableStack<Int32> s7 = s6.Pop();
```

Notice that the Push and Pop methods return a reference to the resulting immutable stack. In contrast, the Push and Pop methods of the mutable Stack<T> return void and T, respectively. This design reflects the fact that changing an immutable stack conceptually results in a whole different stack, while changing a mutable stack actually changes the stack. If the same sequence of operations is performed on a mutable stack, then you'll get a different end result, as shown in **Figure 1**. What makes the immutable stack immutable is that there's no way to change the pointers and values of nodes.

Note that the empty immutable stack node stores the default value of T, which is 0 for Int32. Also, the mutable stack only sets the values of popped items to the default value rather than shrinking the size of the array. The unoccupied part of the array is called slack space.

To get the item at the top of an immutable stack, you can either use the Peek method or another overload of Push, which has an out parameter through which the item is returned.

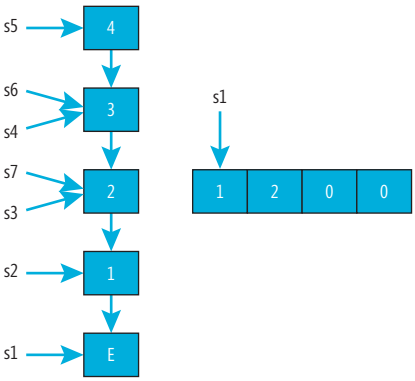


Figure 1 **A Change to an Immutable Stack Results in a Different Stack in Contrast to Mutable Stacks**

Immutable Lists

The list data structure is more complex than the stack mainly due to the indexing operation. The list structure offers item retrieval, addition and removal at a specified index. Using an array, as done in the mutable List<T>, would be reasonable, but, as explained earlier, would be inefficient for a general-purpose immutable list. Using a linked list is also not suitable because you have to potentially traverse many elements to reach the item at a specific index. Instead, balanced binary trees let you implement all operations with respectable performance. Most immutable collections are implemented using balanced binary trees. The rest use linked lists and only one, namely the immutable array, uses arrays as explained in the next section.

Every node in the tree contains an item of the list and, therefore, has an index. The ImmutableList<T> type organizes the tree such that a depth-first, in-order traversal of the tree corresponds to a traversal of the list in order from the item at index 0 to the last item.

Consider the following program:

```
ImmutableList<Int32> l1 = ImmutableList.Create<Int32>();
ImmutableList<Int32> l2 = l1.Add(1);
ImmutableList<Int32> l3 = l2.Add(2);
ImmutableList<Int32> l4 = l3.Add(3);
ImmutableList<Int32> l5 = l4.Replace(2, 4);
```

Figure 2 shows what happens to the underlying binary tree while performing the sequence of operations starting with the empty immutable list. Each box represents a node in the tree. The box that contains the letter E represents the empty tree singleton (the arrows that point nowhere are to be interpreted as pointing to the E box). The boxes and arrows on the right side of the figure are immutable while those on the left are temporarily mutable. This is indicated by an internal Boolean flag called frozen. This flag serves two purposes, as I'll explain next.

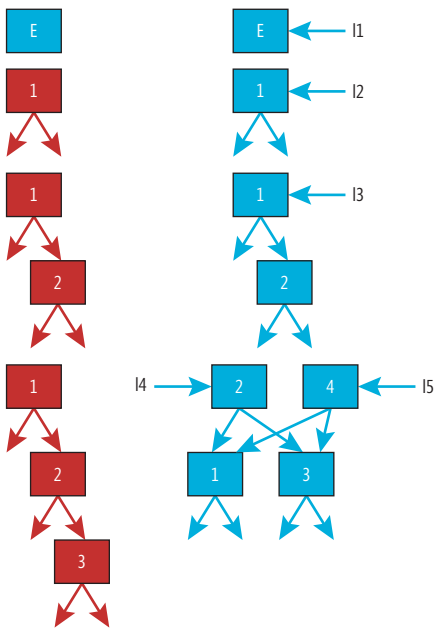


Figure 2 **Internal State of Trees (Left) and the Publically Accessible State Resulting After Completing the Mutations (Right)**

To add the first item to the tree, a new node is created with both of its pointers pointing to the empty node. All newly created nodes start with the frozen flag set to false (temporarily mutable). At this point, nothing more needs to be done and, therefore, the tree is made immutable by setting the frozen flag to true as indicated on the right side of the figure. This makes the tree immutable for the rest of its lifetime.

To add a second item, due to the way the tree is organized, its node has to be the right child of the first node. But because that node is immutable, you can't change its pointers. The only way to add the second item is to not only create a node for it, but also create another node for the first item. That's why l2 and l3 will point to totally separate trees.

Similarly, the third item has to be the right child of the second node. The only way to add it is by creating new nodes for both

the first and second items. However, this time the resulting tree is imbalanced. This situation occurs when the difference between the height of the left and right subtrees of the root is at least 2. To keep the tree balanced, you have to reorganize it so that it becomes the tree shown in the bottom right of **Figure 2**. You can do this because the tree is still mutable and no code outside the ImmutableList<T> type can access it or observe any of the mutations. Before a reference to the tree is returned, it's frozen by setting the frozen flag of each node to true to make it immutable. This demonstrates the first purpose of the frozen flag.

The last line of code calls the Replace function, which finds the specified item and replaces it with another item. In this case, a new node is created to hold the new item and the other nodes of the same tree are reused in the new tree.

Because of the way the tree is organized, any single operation on the list ,whether its addition, insertion, removal or search has a time complexity of O(log N) where N is the number of items currently in the list. In contrast, operations on the mutable list List<T> are either O(1) (where the operation can be performed in place) or O(N) (where the underling array needs to be copied).

You can quickly perform a single operation on an immutable list. But if you want to perform a large number M of operations consecutively, it will take O(M log N). You can do better by taking advantage of the frozen flag and lumping together all mutations. This optimization is offered by Builders.

Most immutable collections including ImmutableList<T> define types called builders that use the same underlying data structures and offer the same APIs. The difference is that a builder doesn't set the frozen flag after every operation. It keeps any new nodes that have been created in a mutable state so that you can perform many operations more efficiently. The builder type for the immutable list is ImmutableList<T>.Builder.

To create a builder instance, you need an immutable collection instance. You can start with an empty collection and use the ImmutableList.CreateBuilder<T> static method or use the ToBuilder instance method on a given immutable collection. In the latter case, all existing nodes will remain, as promised, immutable. It's only those new nodes that will be mutable. Once all operations are performed, you can call the ToImmutable instance method to freeze all nodes, effectively making the collection immutable. ImmutableList<T> provides several instance methods, such as AddRange and RemoveRange, that take a reference to IEnumerable<T> and use a Builder internally to perform the operation on the specified items.

Some operations don't benefit from Builders and they're inherently expensive. The Reverse instance method has to copy all non-leaf nodes in the tree to reverse the order of the items. The Sort instance method is implemented by copying all items to an array, sorting the array and then creating the immutable list from the sorted array.

The mutable List<T> uses an array internally. Inserting or removing items from the middle of the array requires creating a new array and copying all other items. Also allocating extremely large arrays may not be possible in a fragmented address space. The mutable LinkedList<T> solves both of these problems. ImmutableList<T> offers the advantages of both, but makes other operations less efficient. ImmutableList<T> is the immutable collection that corresponds to both List<T> and LinkedList<T>. Note that StringBuilder is essentially List<Char>.

Immutable Arrays

The ImmutableArray<T> type, like ImmutableList<T>, implements an immutable list, but in a different way. ImmutableArray<T> is just a thin wrapper around an array of type T[ ]. It's "thin" because it's a value type that contains a single reference-type field. The array itself is allocated from the managed heap. To perform any mutating operation, a copy is made of the whole array and the operation is performed on it. From this point of view, ImmutableArray<T> is a generalization of String, as it can represent strings of items of any type, not just Char.

All mutating operations take O(N) time using ImmutableArray<T>, but O(log N) time using ImmutableList<T>. However, ImmutableArray<T> is superior in three ways. First, it takes O(1) time to access an item given its index using ImmutableArray<T>, while O(log N) time using ImmutableList<T>. Second, while both implementations offer linear-time iteration, ImmutableArray<T> is cache-friendly because all items are stored consecutively. Iterating over an ImmutableArray<T> can be many times faster than iterating over an ImmutableList<T>. Third, ImmutableArray<T> consumes less memory because it doesn't use pointers. In general, you might need to measure to determine which one to use in a particular situation. Both types implement the IImmutableList<T> interface. You can use this interface across your code to easily switch between the types.

As always, you can improve performance and reduce garbage collection (GC) pressure by performing bulk operations and pooling the Builder objects. You can either use the bulk operations methods XxxRange or ImmutableArray<T>.Builder, which is implemented similarly to List<T>.

Note that because the standard design of LINQ operators works on IEnumerable<T> references, applying them on the value type ImmutableArray<T> requires boxing. The immutable collections NuGet package includes an implementation of some LINQ operators designed specifically for ImmutableArray<T> to avoid boxing. It can be found in System.Linq.ImmutableArrayExtensions.

Immutable Dictionaries

The ImmutableDictionary<TKey, TValue> type uses a balanced binary tree to represent the dictionary. Each node in the tree contains an ImmutableList<KeyValuePair<TKey, TValue>> (which is also a balanced binary tree) that holds all the items that hash to the same value. On the other hand, the mutable Dictionary<TKey, TValue> uses an array of key-value pairs with open addressing for collision resolution. Overall, ImmutableDictionary<TKey, TValue> is many times slower and consumes much more memory than Dictionary<TKey, TValue>. Using the dictionary builder only helps a little because the underlying structure is still a tree of trees. You should definitely measure performance when using ImmutableDictionary<TKey, TValue> to determine whether it's acceptable or not. If it's not, you might need to write your own customized immutable dictionary.

Figure 3 **Amount of Time and Memory Used by Mutable Lists, Immutable Lists and Immutable Arrays**

	Mutable	ImmutableList	ILBuilder	ImmutableArray	IABuilder
Append	0.2	12	8	Horrible!	0.2
Prepend	Horrible!	13.3	7.4	Horrible!	Horrible!
32-bit size	128	320	320	128	128
64-bit size	128	480	480	128	128

Performance and Memory Consumption of Immutable Collections

Now, even if using an immutable collection is principally ideal, it might not result in acceptable performance. That's why it's important to understand how they're implemented and their impact on performance.

Let's compare the performance of the immutable list against that of the mutable counterpart. You can find the time complexities of common operations on immutable collections at bit.ly/2ko07HS. This stuff doesn't say much and it's better to get a little more practical. I've written three programs that simply append or prepend 10 million 8-byte integers to a mutable list, an immutable list and an immutable list builder, respectively. **Figure 3** shows the results. (The numbers shown are approximate. Time is in seconds. Memory is in megabytes. JIT optimizations are enabled. The default constructor is used to create the mutable list.)

Appending to a mutable list is cheap because it's based on an array. The array is doubled in size occasionally, and after that, adding an item is a fast operation. On the other hand, adding an item to an immutable list is far more expensive because it's based on a tree. Even when using the builder, it's still about 40 times slower. That's a huge difference. However, this isn't exactly a fair comparison. A fair comparison would be between the immutable list and a mutable list that uses thread synchronization to provide similar snapshot semantics. Still, this makes immutable collections much less attractive in single-threading scenarios. Even though the time complexity is only O(log N), the hidden constant factor is quite large. I'll explain why shortly.

It's a whole different story for the prepend operation. It would take List<T> many hours to finish because it has to allocate and copy 10 million short-lived arrays of increasingly larger sizes. The immutable list and its builder maintained more or less the same performance.

Figure 4 shows part of the managed memory allocation graph generated using the Diagnostic Tools of Visual Studio 2015 while appending items to an immutable list. Markers at the top of the graph indicate recorded GCs of any generation. The graph shows that GCs happen frequently, about every several dozen milliseconds. I used PerfView to determine the total amount of time spent doing all those little GCs. Without using the builder, the GC time is about 4 seconds. This is the difference between using the immutable list directly and using the builder. To determine whether this difference is indeed due to GCs or whether it's just a coincidence, I also used PerfView on the program that uses the builder. It turns out that this is indeed the case. This can be easily explained by looking at how each works. The immutable list creates many short-lived and medium-lived objects, while the builder mutates the pointers of existing objects. Using the immutable list directly triggered more than 100 GCs while using the builder and the mutable list triggered less than 10.

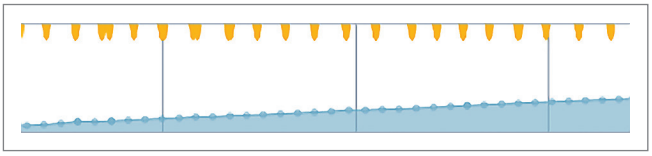


Figure 4 **The GC Runs Much More Often When Appending Items to the Immutable List**

The builder is so much slower than the mutable list. There are four interlinked reasons for this. First, it uses a linked structure that incurs many cache misses. Second, after appending any two items, the tree becomes imbalanced and requires a rotation to rebalance it. Third, appending an item requires traversing (log N) nodes. Fourth, every time an item is added, it triggers a separate memory allocation for the hosting node.

This doesn't mean that the GC is part of the problem. On the contrary, automatic memory management actually makes it a lot easier to write and use immutable collections. It automatically cleans up all those immutable objects that nobody is using.

Let's compare also the immutable stack with the mutable stack. Such comparison lets you quantify the cost of object allocations and their associated cache misses (which are only a small part of the total cache misses that may occur later) resulting from using linked-data structures. The immutable stack is GC-friendly because it only allocates objects that will be part of the resulting stack. It's so efficient that it doesn't even have a Builder. Pushing 10 million 8-byte integers onto a mutable stack takes about 0.17 seconds, while pushing the same onto an immutable stack takes about 1 second. That's about a 6x slowdown, which isn't bad. Iterating over a large immutable stack or any linked structure can be many times slower compared to iterating over the corresponding mutable collection primarily because of cache misses and page faults (and transfers across NUMA nodes on NUMA architectures because of sharing).

That said, the array-based mutable collections suffer from the slack space wastage. If you remove most items from a large collection, the underlying array will not shrink and continue to unnecessarily occupy memory. Linked-based immutable collections always maintain one object per item. However, this is hardly an advantage for linked collections, considering typical use cases.

When to Use Immutable Collections

The fundamental benefit of immutability is that it makes it significantly easier to reason about how the code works, enabling you to quickly write correct and elegant code. Consider a single-threaded program. At a given line of code, you may be wondering about the state of an immutable collection. You can easily determine that by locating those locations in code where the collection was created. There's usually only one of few such locations. By continuing this process, you'll get to either a mutable source or empty the collection. It doesn't matter if the immutable collection has been passed to methods because its structure is guaranteed to be preserved, so you

don't have to consider what happens in those methods. If the items were of an immutable type, as well, you'll be able to reason about the full state of the collection. It's equally easy in multi-threaded programs, but becomes far more difficult when using shared mutable collections. Therefore, immutability can be a general design principle as it is in functional programming.

Now, consider the following method signature:

```
void Foo<T>(Stack<T> s);
```

This method has a mutable stack parameter, so the stack could be modified by the method. This could, indeed, be the purpose of the method. But when it does modify the stack, the old state is lost unless the caller made a copy of it. Note that the method doesn't have to return anything even if it modified the stack. One more thing that this signature conveys is that it doesn't offer any thread-safety guarantees.

This is fine if thread safety isn't a concern and if the method is expected to modify the stack and you're interested in these modifications. If the purpose of the method is to only read or inspect the stack (or it might modify it, but the caller never cares about its modifications), then this signature might be more suitable:

```
void Foo<T>(ReadOnlyCollection<T> s);
```

There are two issues with this. First, ReadOnlyCollection<T> requires a List<T> to be constructed, so the caller has to copy the stack to a list. Making the parameter to be of the interface type IReadOnlyCollection<T> avoids this issue because Stack<T> implements it, but then the method could convert it back to Stack<T> just as easily. Second, if the method usually makes changes to the stack, it has to first copy it to a mutable collection. This signature also doesn't offer any thread-safety guarantees. It's only convenient when the original mutable collection is List<T> and the method doesn't change it.

In scenarios where there are potentially multiple threads accessing the collection, this signature might be more suitable:

```
void Foo<T>(ConcurrentStack<T> s);
```

The concurrent stack is mutable, so all threads will see all the changes. This is only suitable when at least one of the following two conditions occur: Either the method is expected to consider changes potentially made by other threads as soon as they're made, or other threads want to see the changes made by the method as soon as they're made. Note that any thread can see any specific change separately. Otherwise, if some threads should only see a batch of changes or none of the changes, they have to acquire a global lock and make a private copy of the stack. These two situations are called snapshot semantics.

Notice how in various scenarios, different collection types need to be used. Also a good documentation would be desirable to explain how the method works or how it should be used. Immutable collections simplify all of this. The following two signatures satisfy most scenarios:

```
void Foo1<T>(ImmutableStack<T> s);
ImmutableStack<T> Foo2<T>(ImmutableStack<T> s);
```

Look how beautiful these APIs are. The first signature is used when no one cares about the changes made by the method. Otherwise, the second signature can be used. There are only two situations in which immutable collections are unsuitable: throughput (the rate at which items are processed) and producer-consumer semantics. As demonstrated earlier, the general-purpose immutable collections have inferior throughput, particularly in single-threaded scenarios. In producer-consumer semantics, concurrent mutable collections are more elegant and result in better performance. The difference between producer-consumer semantics and snapshot semantics is in how the reading or consuming agents behave. In the former, elements get consumed (removed permanently) while in the latter, elements get processed and would only be removed by the writing or producing agents. I would like to refer to snapshot semantics as changer-processor semantics because there are changing agents and processing agents. Processing agents can make changes as long as these changes are kept in a separate copy that's needed along with other copies. If these changes were to be made global, you would be in the realm of producer-consumer semantics.

Convenient Interfaces and APIs

There are many ToXxx extension methods that convert from mutable collections or collections-related interfaces to immutable collections. These methods copy the mutable collection rather than reuse it to maintain immutability. Many immutable collections offer convenient methods, such as those for sorting and searching, similar to those offered by the mutable ones. This helps mixing code that uses both kinds of collections.

To make immutable collections more usable in existing code bases, some of them implement suitable common interfaces such as IEnumerable<T>, IList<T> and ICollection<T>. Some of the methods declared such as IList<T>.Insert are intended to mutate the collection. These are implemented by throwing a NotSupportedException. Immutable collections also implement the corresponding immutable interfaces that are defined in the NuGet package.

The System.Collections.Immutable.ImmutableInterlocked type offers a number of methods that implement interlocked exchange mechanisms to correctly update items in or references to immutable collections. For example, the following method takes a reference to an item and updates it according to the specified transformer:

```
public static bool Update<T>(ref T location, Func<T, T> transformer) where T : class;
```

While the effects of such operations are observable by all threads, it guarantees that the same item will always be observed by all of them.

Wrapping Up

This article discussed the benefits of immutable collections and provided a detailed discussion of the design and implementation of immutable stacks, lists, arrays and dictionaries. There are numerous other immutable collections shipped with the NuGet package. Almost every mutable collection has a corresponding immutable one, which you can find there. I hope that you can effectively use these types in your code. If you like the immutability pattern and would like to write your own immutable types, Andrew Arnott has written a Roslyn-based tool that makes writing immutable types as easy as writing mutable types. You can find more about it at bit.ly/2ko2s50. ■

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

Thanks to the following technical experts for reviewing this article: *Andrew Arnott and Immo Landwerth*

Extreme Performance Linear Scalability

NCache™

Distributed Cache

- In-Memory App Data Caching
- ASP.NET Sessions & View State
- Runtime Data Sharing

NosDB

NoSQL Database

- Schema-Free JSON Documents
- Multiple Shards & Data Replication
- Powerful SQL, LINQ, ADO.NET

100% Native .NET

Open Source (Apache 2.0 License)

Microsoft Azure amazon web services™ Windows Server 2012 Visual Studio

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

JAMES MCCAFFREY

Chi-Squared Goodness of Fit Using C#

A chi-squared (also called chi-square) goodness of fit test is a common statistical technique that's used to determine if observed-count data matches expected-count data. For example, suppose you have three Web server machines designed to handle 50 percent, 30 percent and 20 percent of your traffic, respectively. If you observe 1,000 HTTP requests, you'd expect to see 500 requests handled by the first server, 300 requests handled by the second server and 200 requests handled by the third server.

But suppose your actual observed counts are (480, 290, 230). Do you decide that the differences between observed and expected counts are simply due to randomness, or do you conclude there's statistical evidence that your Web servers aren't handling traffic as intended? This is an example of the type of issue a chi-squared goodness of fit test can address.

A good way to see where this article is headed is to take a look at the demo program in **Figure 1**. The demo program analyzes an American-style roulette wheel. This is a standard chi-squared goodness of fit example. An American roulette wheel has 38 slots, where 18 are red, 18 are black, and 2 are green. So if you spin the wheel 380 times, and the wheel is fair in the sense that every slot is equally likely, you'd expect to see 180 reds, 180 blacks, and 20 greens.

```
file:///C:/ChiSquaredUsingCSharp/bin/Debug/ChiSquaredUsingCSharp.EXE

Begin chi-squared goodness-of-fit demo

Goal is to determine if roulette wheel is fair

Wheel was spun 380 times

Observed counts of red, black, green:
192  163  25

Probabilities if fair:
0.4737  0.4737  0.0526

Expected counts if fair:
180.0  180.0  20.0

Calculated chi-squared = 3.66
The pval with df of 2 = 0.1608

The pval is approximate probability that, if wheel is fair,
you'd see a chi-squared value as extreme as calculated

End demo
```

Figure 1 **Chi-Squared Goodness of Fit Demo**

> A chi-squared goodness of fit test is a common statistical technique that's used to determine if observed-count data matches expected-count data.

But suppose you observed counts of (192, 163, 25)—a few more reds and greens than expected, and fewer blacks than expected. Using the observed and expected counts, the demo program calculates a chi-squared statistic of 3.66. This value is a measure of how different the observed and expected counts are, where larger values indicate greater difference.

The demo then uses the calculated chi-squared statistic to compute a p-value ("probability value") of 0.1608. This value is the approximate probability you'd see a calculated chi-squared value of 3.66 or greater if in fact the roulette wheel is fair. Put another way, the smaller the p-value is, the more likely the wheel isn't fair. Here, the p-value of 0.1608 is inconclusive. The probability that the wheel is fair is 0.1608, so the probability that the wheel isn't fair is 1 - 0.1608 = 0.8392 which is quite high, but not decisive.

This article assumes you have intermediate or higher programming skills but doesn't assume you know anything about chi-squared goodness of fit tests. The demo program is coded using C#, but you should have no trouble refactoring the code to another language, such as JavaScript or Python. All the demo code is presented in this article and is also available in the accompanying download.

Code download available at msdn.com/magazine/0317magcode.

The Demo Program Structure

The overall structure of the demo program, with a few minor edits to save space, is presented in **Figure 2**. I use a static method style rather than an object-oriented programming style for simplicity.

TECHMENTOR

IN-DEPTH TRAINING FOR IT PROS

AUGUST 7 - 11, 2017

MICROSOFT HEADQUARTERS

REDMOND, WA

PLUG IN TO NEW KNOWLEDGE @ THE SOURCE

WHAT SETS TECHMENTOR APART?

+ Immediately usable IT education
+ Training you need today, while preparing you for tomorrow
+ Zero marketing-speak, a strong emphasis on doing more with the technology you already own, and solid coverage of what's just around the corner
+ Intimate setting, where your voice is heard, making it a viable alternative to huge, first-party conferences
+ Experience life @ Microsoft Headquarters for a full week

YOU OWE IT TO YOURSELF, YOUR COMPANY AND YOUR CAREER TO BE AT TECHMENTOR REDMOND 2017!

HOT TRAINING TOPICS INCLUDE:

+ Windows Server + Hyper-V + Windows PowerShell + DSC + DevOps + Azure + Security + And More! +

++

REGISTER NOW

SAVE $400
USE PROMO CODE TMMAR1

[TECHMENTOREVENTS.COM/REDMOND]

SUPPORTED BY: Redmond · Redmond Channel Partner · Virtualization Review

PRODUCED BY: 1105 MEDIA

Figure 2 **Chi-Squared Demo Program Structure**

```
using System;
namespace ChiSquaredUsingCSharp
{
  class ChiSquaredProgram
  {
    static void Main(string[] args)
    {
      Console.WriteLine("Begin demo \n");

      // 1. Calculate chi-squared stat.
      // 2. Use chi-squared to calculate p-value.

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

    public static void ShowVector(int[] v) { . . }

    public static void ShowVector(double[] v,
      int dec) { . . }

    public static double ChiFromFreqs(int[] observed,
      double[] expected) { . . }

    public static double ChiFromProbs(int[] observed,
      double[] probs) { . . }

    public static double[] ExpectedFromProbs(
      double[] probs, int N) { . . }

    public static double ChiSquarePval(double x,
      int df) { . . }

    private static double Exp(double x) { . . }

    public static double Gauss(double z) { . . }

  } // Program
} // ns
```

The Main method has all the control logic. The demo program isn't quite as complicated as it might first appear because most of the nine methods are short helpers.

To create the demo program, I launched Visual Studio and created a new C# console application program and named it ChiSquaredUsingCSharp. I used Visual Studio 2015, but the demo program has no significant .NET dependencies so any recent version of Visual Studio will work.

After the template code loaded into the editor window, I right-clicked on file Program.cs in the Solution Explorer window and renamed the file to ChiSquaredProgram.cs, then allowed Visual Studio to automatically rename class Program for me. At the top of the template-generated code, I deleted all unnecessary using statements, leaving just the one that references the top-level System namespace.

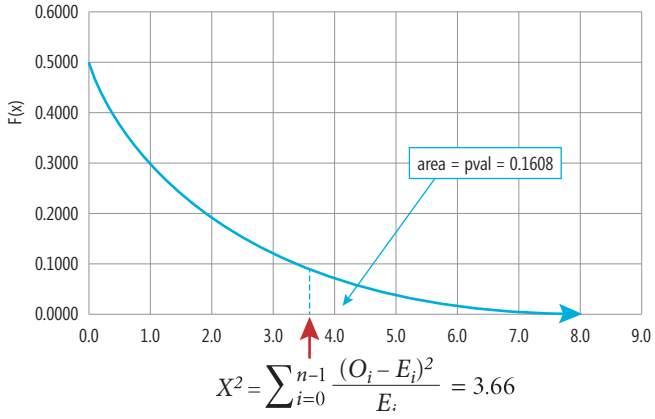


Figure 3 **The Chi-Squared Distribution for df = 2**

The Main method sets up the observed counts of the roulette wheel like so:

```
Console.WriteLine("Is roulette wheel fair");
Console.WriteLine("Wheel was spun 380 times");
int[] observed = new int[] { 192, 163, 25 };  // 380
Console.WriteLine("Observed counts red, black, green:");
ShowVector(observed);
```

I just made these observed counts up because they give a representative example. Helper method ShowVector that displays an integer array is defined:

```
public static void ShowVector(int[] v)  {
  for (int I = 0; i < v.Length; ++i)
    Console.Write(v[i] +"" "");
  Console.WriteLine""\"");
}
```

Next, instead of setting up the expected counts directly, I set them up indirectly like this:

```
double[] probs = new double[] {
  18.0/38, 18.0/38, 2.0/38 };
Console.WriteLine("Probabilities if fair:");
ShowVector(probs, 4);

double[] expected = ExpectedFromProbs(probs, 380);
Console.WriteLine("Expected counts if fair:");
ShowVector(expected, 1);
```

The probabilities of red, black, and green are 18/38, 18/38, and 2/38, as explained earlier. Helper method ExpectedFromProbs accepts an array of probabilities and a total count, and returns an array of expected counts. I could have directly set up the expected counts for 380 spins of the roulette wheel like this:

```
double[] expected = new double[] { 180.0, 180.0, 20.0 };
```

The expected-count helper method is defined:

```
public static double[] ExpectedFromProbs(double[] probs,
  int N)
{
  double[] expected = new double[probs.Length];
  for (int i = 0; i < probs.Length; ++i)
    expected[i] = probs[i] * N;
  return expected;
}
```

And the overloaded ShowVector method for an array of type double is defined:

```
public static void ShowVector(double[] v, int dec) {
  for (int i = 0; i < v.Length; ++i)
    Console.Write(v[i].ToString("F" + dec) + "  ");
  Console.WriteLine("\n");
}
```

The demo program continues by calculating the chi-squared statistic:

```
double chi = ChiFromProbs(observed, probs);
Console.WriteLine("Calculated chi-squared = " +
  chi.ToString("F2"));  // 3.66
```

Method ChiFromProbs uses a signature that accepts an integer array of observed counts and a double array of expected probabilities, mostly because that's the signature used by the equivalent R language chisq.test function. For example, in an interactive R shell you could perform the demo like so:

```
> obs <- c(192, 163, 25)
> probs <- c(18/38, 18/38, 2/38)
> chisq.test(x=obs, p=probs)

X-squared = 3.6556, df = 2, p-value = 0.1608
```

Notice the calculated chi-squared statistic (3.66) and p-value (0.1608) obtained using R are the same values as computed by

the demo program. The demo concludes by using the calculated chi-squared statistic to compute the p-value:

```
int df = observed.Length - 1;
double pval = ChiSquarePval(chi, 2);
Console.WriteLine("The pval with df of " + df +
  " = " + pval.ToString("F4") );
Console.WriteLine("Pval is probability, if wheel fair,");
Console.WriteLine("you'd see a chi-squared as calculated");
Console.WriteLine("End demo");
Console.ReadLine();
```

Variable df stands for degrees of freedom, which I'll explain shortly.

Understanding the Chi-Squared Statistic

If you have an array of observed counts and an array of expected counts, you can calculate a metric called the chi-squared statistic, a measure of how different the observed and expected counts are. Larger values indicate greater difference.

The chi-squared statistic is defined as the sum of the squared differences between observed and expected divided by expected:

chi-squared = sum((obs[i] - exp[i])^2 / exp[i])

The idea is best explained by an example. Suppose, as in the demo, the observed counts for 380 spins of a roulette wheel are (192, 163, 25) and the expected counts if the wheel is fair are (180, 180, 20). The calculated chi-squared statistic is:

```
chi-squared = (192 - 180)^2 / 180 +
              (163 - 180)^2 / 180 +
              (25 - 20)^2   / 20

            = (144 / 180) + (289 / 180) + (25 / 20)
            = 0.8000 + 1.6056 + 1.2500
            = 3.6556
```

Figure 4 **Methods ChiSquarePval and Gauss**

```
public static double ChiSquarePval(double x, int df)
{
  // x = a computed chi-square value.
  // df = degrees of freedom.
  // output = prob. x value occurred by chance.
  // ACM 299.

  if (x <= 0.0 || df < 1)
    throw new Exception("Bad arg in ChiSquarePval()");

  double a = 0.0; // 299 variable names
  double y = 0.0;
  double s = 0.0;
  double z = 0.0;
  double ee = 0.0; // change from e
  double c;

  bool even; // Is df even?

  a = 0.5 * x;
  if (df % 2 == 0) even = true; else even = false;

  if (df > 1) y = Exp(-a); // ACM update remark (4)

  if (even == true) s = y;
  else s = 2.0 * Gauss(-Math.Sqrt(x));

  if (df > 2)
  {
    x = 0.5 * (df - 1.0);
    if (even == true) z = 1.0; else z = 0.5;
    if (a > 40.0) // ACM remark (5)
    {
      if (even == true) ee = 0.0;
      else ee = 0.57236494292470008707171135;
      c = Math.Log(a); // log base e
      while (z <= x) {
        ee = Math.Log(z) + ee;
        s = s + Exp(c * z - a - ee); // ACM update remark (6)
        z = z + 1.0;
      }
      return s;
    } // a > 40.0
    else
    {
      if (even == true) ee = 1.0;
      else
        ee = 0.56418958354775628694807945 / Math.Sqrt(a);

      c = 0.0;
      while (z <= x) {
        ee = ee * (a / z); // ACM update remark (7)
        c = c + ee;
        z = z + 1.0;
      }
      return c * y + s;
    }
  } // df > 2
  else {
    return s;
  }
} // ChiSquarePval()

private static double Exp(double x)
{
  if (x < -40.0) // ACM update remark (8)
    return 0.0;
  else
    return Math.Exp(x);
}

public static double Gauss(double z)
{
  // input = z-value (-inf to +inf)
  // output = p under Normal curve from -inf to z
  // ACM Algorithm #209
  double y; // 209 scratch variable
  double p; // result. called 'z' in 209
  double w; // 209 scratch variable

  if (z == 0.0)
    p = 0.0;
  else
  {
    y = Math.Abs(z) / 2;
    if (y >= 3.0)
    {
      p = 1.0;
    }
    else if (y < 1.0)
    {
      w = y * y;
      p = (((((((0.000124818987 * w
        - 0.001075204047) * w + 0.005198775019) * w
        - 0.019198292004) * w + 0.059054035642) * w
        - 0.151968751364) * w + 0.319152932694) * w
        - 0.531923007300) * w + 0.797884560593) * y
        * 2.0;
    }
    else
    {
      y = y - 2.0;
      p = (((((((((((-0.000045255659 * y
        + 0.000152529290) * y - 0.000019538132) * y
        - 0.000676904986) * y + 0.001390604284) * y
        - 0.000794620820) * y - 0.002034254874) * y
       + 0.006549791214) * y - 0.010557625006) * y
       + 0.011630447319) * y - 0.009279453341) * y
       + 0.005353579108) * y - 0.002141268741) * y
       + 0.000535310849) * y + 0.999936657524;
    }
  }

  if (z > 0.0)
    return (p + 1.0) / 2;
  else
    return (1.0 - p) / 2;
} // Gauss()
```

The demo implements this function as:

```
public static double ChiFromFreqs(int[] observed,
  double[] expected)
{
  double sum = 0.0;
  for (int i = 0; i < observed.Length; ++i) {
    sum += ((observed[i] - expected[i]) *
      (observed[i] - expected[i])) / expected[i];
  }
  return sum;
}
```

There's no error-checking here, for simplicity, but in a production system you'd want to make sure that arrays observed and expected have the same length, and so on. The demo program also has a method to calculate a chi-squared statistic from an array of observed counts and an array of expected probabilities:

```
public static double ChiFromProbs(int[] observed,
  double[] probs)
{
  int n = observed.Length;
  int sumObs = 0;
  for (int i = 0; i < n; ++i)
    sumObs += observed[i];

  double[] expected = ExpectedFromProbs(probs, sumObs);
  return ChiFromFreqs(observed, expected);
}
```

To recap, the math definition of the chi-squared statistic uses an array of observed counts and an array of expected counts. It's also useful to calculate chi-squared from an array of observed counts and an array of expected probabilities. To perform this calculation, it's handy to have a helper method that calculates expected counts from expected probabilities.

Understanding the Chi-Squared Distribution

If you have a calculated chi-squared statistic value, you can use it to calculate the probability (p-value) of getting that chi-squared value. This idea is best explained visually. Take a look at the graph in **Figure 3**, which shows the chi-squared distribution for the demo problem. The total area under any chi-squared distribution is 1.0. The p-value is the area under the graph from the calculated chi-squared statistic to +infinity.

> Complicating matters is the fact that there isn't just one chi-squared distribution, there are many.

This idea is very subtle. And, unfortunately, calculating the area under a chi-squared distribution is very, very difficult. Luckily, numerical algorithms to estimate the area under a chi-squared distribution were developed in the 1960s and 1970s and are still used today.

Complicating matters is the fact that there isn't just one chi-squared distribution, there are many. Briefly, for a goodness of fit test, there's a value called the degrees of freedom, abbreviated df. The value of df for a goodness of fit test is simply 1 minus the number of count buckets. For the demo, there are three count buckets (red, black, green) so the df is 3 - 1 = 2.

There's a different chi-squared distribution for each value of df. The graph in **Figure 3** shows the chi-squared distribution for df = 2. Notice the chi-squared values on the horizontal axis run from 0.0 (the smallest possible calculated chi-squared statistic, when all observed counts equal their corresponding expected counts), to +infinity (there is no limit to how different observed and expected counts can be).

> There are several sophisticated algorithms that can calculate a p-value / area under the chi-squared distribution graph.

There are several sophisticated algorithms that can calculate a p-value / area under the chi-squared distribution graph. The demo program uses what's called ACM Algorithm 299, which is implemented as method ChiSquarePval. The algorithm in turn uses another algorithm called ACM 209, which is implemented as method Gauss. These algorithms are foundations of numerical computing and are presented in **Figure 4**. Even a brief glance at the code should convince you there's some very serious math going on, but luckily you can think of the methods that implement these algorithms as black boxes because you'll never have to modify the code.

Wrapping Up

It's not likely you'll need to analyze roulette wheel data, but once you understand the type of problem scenario where a chi-squared goodness of fit test is applicable, you might find many practical uses for the test. There are many tools that can perform a chi-squared goodness of fit test, including Excel and the R language, but these tools can be difficult or impossible to integrate into a software system. In such situations you can use the code presented in this article.

There are several other types of statistical tests that use the chi-squared distribution. The test presented in this article is sometimes called Pearson's chi-squared test to distinguish it from other chi-squared tests. Also, there are several other statistical tests that can compare observed and expected counts, though chi-squared is the most common.

It's important to remember that the chi-squared goodness of fit test, like most statistical tests, is probabilistic, so you should interpret results conservatively. Even if you get a very small p-value, it's preferable to say something like, "The small p-value of 0.0085 suggests that the difference between observed and expected counts is unlikely to have occurred by chance," rather than, "The small p-value indicates the observed counts couldn't have occurred by chance." ■

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: *Chris Lee and Kirk Olynyk*

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

REDMOND
AUGUST 14-18, 2017
MICROSOFT HEADQUARTERS

JOIN US AT MICROSOFT HEADQUARTERS THIS SUMMER

- Rub elbows with blue badges
- Experience life on campus
- Enjoy lunch in the Commons and visit the Company Store
- Networking event on Lake Washington, Wednesday, August 16
- And so much more!

EVENT PARTNERS
Microsoft

SUPPORTED BY
Visual Studio msdn magazine Visual Studio MAGAZINE

PRODUCED BY
1105 MEDIA

TED NEWARD

How To Be MEAN: Angular Components

Welcome back, MEANers.

Last month I began my descent into Angular—getting started with it, working with the base template, and taking a quick look at some of the more obvious parts that make up the simple Angular "Hello World" application (msdn.com/magazine/mt793274). It's clear that Angular (formerly called Angular 2) takes a slightly different approach to building Single-Page Applications (SPAs) than many other JavaScript Web frameworks, including its immediate predecessor, AngularJS (formerly known as Angular 1). But these differences are not just the result of open source peevishness at wanting to do it differently just for the sake of doing it differently; a new approach to building Web applications is building a groundswell of interest, and Angular has chosen to grab on with both hands.

The new word of the Web is "components."

Components

There are two ways to think about the word components. One is to break down the major parts that make up a traditional Angular application. That list is relatively short and similar to Angular or other Web frameworks:

- Modules
- Components
- Services
- Routes

A few other parts are lurking in the Angular world, but for the most part, these are the four key components that make up an Angular application.

Modules are the top-level unit of segregation and cognition. Essentially, several modules make up an application, where a module is a container around a reasonably coupled group of smaller parts (components, services and so on). Modules are generally often units of deployment—this is what the browser will download to the user's machine in order to execute. More important, they're simply units of cognitive separation in the same way that .NET assemblies are. If all of the code, resources and other pertinent assets pertaining to "viewing and displaying the current time" are grouped together into a single "time" module, developers working with the system don't have to think about time unless they're working with (or in) that module.

This reduces the overall cognitive load on the developer trying to keep track of all the moving parts within the system. Obviously, developers are free to put everything into a single module, just as it's certainly possible to write an entire desktop application in one .NET assembly. However, most developers with any amount of experience under their belts will have some opinions on where those module lines should be drawn, and they'll modularize their code accordingly. (Whether any two developers agree on where those lines are, on the other hand, is an entirely different story.)

Components are like modules, but writ smaller—where a module may contain all the definitions for doing things relating to HTTP communication (such as Angular's HttpModule) or HTML forms (FormsModule), components are typically single-purpose and directly related to the UI. In the Hello World application from last month's column, the code had exactly one component—the AppComponent—which provided a simple, non-interactive text display greeting the world. Despite its simplicity, it was and is a full-blooded Angular component, as evidenced by the fact that the component can be used in a more-or-less opaque fashion, via the <my-app> tag that served as the component's tag. In essence, the tag usage becomes the client "surface area" for using the component.

Modules are generally also often units of deployment—this is what the browser will download to the user's machine in order to execute.

Services, on the other hand, are more like low-level libraries that typically provide access to underlying functionality that shouldn't be a part of the component itself. In an Angular approach, usually making any sort of HTTP API call (such as to the Node/Express/Mongo back end that Microsoft has been hacking out over the last year) should be bundled into a service and simply used from the components that require it. Typically, there will be fewer services than components in a given Angular application, though of course, "your mileage may vary."

Finally, routes are the primary mechanism of navigation. Routes define the mapping of incoming URL patterns to how the Angular application should respond. Similar in spirit to how Express.js routes were written, routes will be defined in a "routing map" for

easy reference from one place, and usually a route will map to a component (though that component in turn might—and often will—make use of other components).

This might seem a little overwhelming, particularly the heavy use of the word "component," but in reality, it's simpler than it seems. Let's start by defining a new component, referenced from the AppComponent application component (sometimes called the "root component" because it's where the action begins) that displays a name and a button that, when clicked, prints that name to the console. It's a trivial component, but it'll serve as an easy way to see how to build a new component, reference it, and use it.

GreetingsComponent

By convention, Angular likes to suffix any component type with the word "component," so because this component is designed to offer greetings, GreetingsComponent seems like an appropriate—if boring—name for it. It will live in a file called greetings.component.ts, but where that file should live is not an automatic decision; some Angular developers prefer to put each component into its own directory, such as greetings, or some will put it into a components directory. On the other hand, of course, you can always just leave it right next to app.component.ts. Debate rages among Angular developers as to which is the superior option, but for simplicity, we'll do this as a peer to app.component.ts, so that greetings.component.ts lives directly inside the app directory.

The opening phrases of greetings.component.ts are remarkably similar to the AppComponent from last time:

```
import { Component } from '@angular/core';

@Component({
  selector: 'greetings',
  template: '<div>Hello</div>'
})
export class GreetingsComponent {
}
```

> Finally, routes are the primary mechanism of navigation.

It's really just a simple label. Nothing complex to see here.

Using it, however, means you have to wire up the GreetingsComponent into the AppComponent, so that the AppComponent's HTML will include it, like so:

```
@Component({
  selector: 'my-app',
  template: `<h1>Hello {{name}}</h1><greetings></greetings>`,
})
export class AppComponent  { name = 'Angular'; }
```

The only change here is in the "template" metadata on the AppComponent. (We'll find a better place for that to live before too long, don't worry; you're not going to have tons of HTML in your component class file.)

If, however, you save the newly modified app.component.ts, Angular will reload (assuming you ran "npm start," or it was still running from last month), and you won't see the "Hello" message. This is because the GreetingsComponent isn't yet loaded, so Angular can't recognize the <greetings> tag as one that's placeholding for a component. To do that, you crack open the app module file, app.module.ts, and register GreetingsComponent as one of your own:

```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppComponent }  from './app.component';
import { GreetingsComponent } from './greetings.component';

@NgModule({
  imports:      [ BrowserModule ],
  declarations: [ AppComponent, GreetingsComponent ],
  bootstrap:    [ AppComponent ]
})
export class AppModule { }
```

The only changes here are to add the "import" that pulls in the GreetingsComponent from disk and to register it as a "declaration" in the @NgModule metadata decorating the AppModule class. Assuming everything is spelled correctly, and the files exist, the text "Hello" appears. Progress!

However, now you need to make some changes to the component itself—and the key here is that the changes are to the component, neatly encapsulated away, and not to anything else.

Adding a Property

First, a generic "Hello" seems lame. Let's add a field to GreetingsComponent to give it at least some level of personalization:

```
import { Component } from '@angular/core';

@Component({
  selector: 'greetings',
  template: '<div>Hello my name is {{name}}.'
})
export class GreetingsComponent {
  constructor() {
    this.name = 'Tarzan'
  }
}
```

As soon as you save the file, you should see "Hello my name is Tarzan" show up. The "name" property is just a standard TypeScript property—nothing magical there.

However, it would be nice if the name could be determined externally, rather than hardcoded into the class as "Tarzan." This is the role of the @Input property:

```
import { Component, Input } from '@angular/core';

@Component({
  selector: 'greetings',
  template: '<div>Hello my name is {{name}}.'
})
export class GreetingsComponent {
  @Input() name : string;
  constructor() { }
}
```

Notice two things: First, you strongly typed the property name so that TypeScript can help ensure that only strings get passed there; second, @Input() is annotated in front of it. This tells Angular that the actual data for the property should come from the tag usage in a parent component, which you now amend in app.component.ts to look like:

```
@Component({
  selector: 'my-app',
  template: `<h1>Hello {{name}}</h1><greetings name="Tarzan"></greetings>`,
})
export class AppComponent  { name = 'Angular'; }
```

Notice that "name" is overloaded here; name is a property used inside of AppComponent, but it's also a property of GreetingsComponent, and the two are clearly separate. Naturally, if passing

the name from AppComponent to GreetingsComponent is the goal, that's accomplished by using the {{name}} interpolation binding within GreetingsComponent's template rather than the literal "Tarzan."

In many cases, the input passed into a component will be something that's not a string; in those situations, the syntax for a property binding like this takes on a slightly different cast, as in:

```
<user-profile [user]="currentUser"></user-profile>
```

In many cases, the input passed into a component will be something that's not a string.

Here, the user property is of a non-string type (probably something like UserProfile), and currentUser is the currently authenticated user. You'll see more of this syntax later, when you start working with non-primitive types (like a Speaker class) to hold the data used by a component.

Adding a Method

Angular components are, at their heart, just TypeScript classes, so it's easy to add a new method to the class that "does something," such as print the name to the console:

```
@Component({
  selector: 'greetings',
  template: '<div>Hello my name is {{name}}.</div>'
})
export class GreetingsComponent {
  @Input() name : string;
  constructor() { }
  sayMyName() {
    console.log('My name is', this.name)
  }
}
```

The sayMyName method does exactly that, just printing the name property to the browser console. However, often these methods will want to be invoked via user action—that means that they have to be wired up to browser events, and that requires another, slightly different, syntax to bind the method to an Angular event, such as a button click. So if the GreetingsComponent defines a button as part of its HTML template, then it can bind the button's click event to call sayMyName, like so:

```
@Component({
  selector: 'greetings',
  template: '<div>Hello my name is {{name}}.' +
    ' <button (click)="sayMyName()">Say my name</button></div>'
})
export class GreetingsComponent {
  @Input() name : string;
  constructor() { }
  sayMyName() {
    console.log('My name is', this.name)
  }
}
```

Event-binding syntax looks similar to traditional HTML event binding, except that Angular insists that event binding must be parenthesized (so it's (click) instead of onClick); property bindings use square brackets, and event bindings use round brackets. It's a little odd to see at first, but over time it starts to become more comfortable, and dare I say convenient, because now the symbols make it more clear which are properties and which are events.

Removing the HTML

If the thought of the HTML living inside of TypeScript code feels a little odd, the Component lets the template live in an external file, usually following the naming convention of component.html to go alongside component.ts. Therefore, for example, if the GreetingsComponent wants to keep its entire HTML in a standalone file, it goes into greetings.component.html, and is referenced using the templateUrl attribute in the @Component metadata on the component itself:

```
@Component({
  selector: 'greetings',
  templateUrl: './app/greetings.component.html'
})
export class GreetingsComponent {
  @Input() name : string;
  constructor() { }
  sayMyName() {
    console.log('My name is', this.name)
  }
}
```

Pay careful attention to the URL used in templateUrl because it's relative to the root of the application as a whole, so even though greetings.component.html sits right next to greetings.component.ts, the HTML file must be referenced from the parent of the "app" directory.

Wrapping Up

Angular's approach to building a Web application starts to become clearer: Developers are expected to define smaller-grained components, and then combine them in various ways to form larger-scale applications. It's certainly not a new concept; in fact, it dates back to the good old days of Visual Basic 3 and the other freewheeling tools of the '90s. The Angular team has simply brought those concepts forward, into the Web world, and combined them with some more modern tooling to make the experience a little nicer and more Web-like. Certainly, a developer could define the entire application as a single component, but it won't take long before the pain of trying to do this will vastly outweigh the gain (if there is any—I'm not convinced there would be).

Much remains to be done, but some of the path begins to make it clear: If this application is going to be a database of speakers, then we'll need to define some SpeakerComponents, at a minimum, to do the display and editing of speakers. Communicating that to the back end will require a service or two. Putting some decorations and lights around the edges will probably be a few more components, and then some routes to have some URLs that people can use. Hang tight, there's much more yet to come. In the meantime, happy coding! ■

Ted Neward *is a Seattle-based polytechnology consultant, speaker and mentor. He has written more than 100 articles, is an F# MVP, 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: *Ward Bell*

Spreadsheets Made Easy.

Fastest Calculations

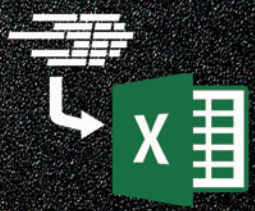
Evaluate complex Excel-based models and business rules with the fastest and most complete Excel-compatible calculation engine available.

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

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

LAS VEGAS
MAR 13-17 2017
BALLY'S, LAS VEGAS, NV

EVENT PARTNERS

Microsoft

Magenic

PLATINUM SPONSORS

ActivePDF®

Alachisoft®

SUPPORTED BY

Visual Studio

INTENSE TRAINING FOR DEVELOPERS, ENGINEERS, PROGRAMMERS, ARCHITECTS AND MORE!

LAST CHANCE!

Development Topics include:

- Visual Studio / .NET Framework
- JavaScript / HTML5 Client
- Modern App Development
- Mobile Client
- Software Practices
- Database and Analytics
- Angular JS
- ASP.NET / Web Server
- Agile
- ALM / DevOps
- Cloud Computing
- Windows Client

TURN THE PAGE FOR FULL AGENDA DETAILS

Sunday Pre-Con Hands-On Labs

Choose From:

- Angular
- Azure
- XAML

NEW! ONLY $695

SPACE IS LIMITED

Register by March 13 and Save $300!*

MUST use discount code LVEB01

*Available on 3 and 5 day packages only.

REGISTER NOW

ROCK YOUR CODE TOUR 2017

LAS VEGAS	AUSTIN	WASH. DC	REDMOND	CHICAGO	ANAHEIM	ORLANDO
MARCH 13-17	MAY 15-18	JUNE 12-15	AUG 14-18	SEPT 18-21	OCT 16-19	NOV 13-17

msdn magazine | Visual Studio MAGAZINE

PRODUCED BY 1105 MEDIA INC. YOUR GROWTH. OUR BUSINESS.

vslive.com/lasvegasmsdn

Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS

AGENDA AT-A-GLANCE

Bally's Hotel & Casino will play host to all Visual Studio Live! Las Vegas sessions, activities, and events!

CONNECT WITH VISUAL STUDIO LIVE!

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

REGISTER NOW

ALM / DevOps | **Cloud Computing** | **Database and Analytics** | **Mobile Client** | **Software Practices**

START TIME	END TIME	
		Full Day Hands-On Labs: Sunday, March 12, 2017 *(Separate entry fee required)*
7:30 AM	9:00 AM	Pre-Conference Workshop Registration - Coffee and Morning Pastries
9:00 AM	1:00 PM	**HOL01** Full Day Hands-On Lab: Build an Azure App in a Day - *Brian Randell*
1:00 PM	2:00 PM	Lunch @ Le Village Buffet, Paris Las Vegas
2:00 PM	6:00 PM	Hands-On Lab Continues
		Pre-Conference Workshops: Monday, March 13, 2017 *(Separate entry fee required)*
7:30 AM	9:00 AM	Pre-Conference Workshop Registration - Coffee and Morning Pastries
9:00 AM	1:00 PM	**M01** Workshop: Native Mobile App Development for iOS, Android and Windows Using C# - *Marcel de Vries & Roy Cornelissen*
1:00 PM	2:00 PM	Lunch @ Le Village Buffet, Paris Las Vegas
2:00 PM	6:00 PM	Workshop Continues
7:00 PM	9:00 PM	Dine-A-Round
		Day 1: Tuesday, March 14, 2017
8:00 AM	9:00 AM	**Keynote:** Rub DevOps On It - *Donovan Brown, Senior DevOps Program Manager,*
9:15 AM	10:30 AM	**T01** Essential Web Development with ASP.NET Core - *Mark Michaelis* / **T02** An Overview of the Xamarin Programming Platforms - *Laurent Bugnion*
10:45 AM	12:00 PM	**T06** Migrating to ASP.NET Core - A True Story - *Adam Tuliper* / **T07** Building Truly Universal Applications with Windows, Xamarin and MVVM - *Laurent Bugnion*
12:00 PM	1:00 PM	Lunch
1:00 PM	1:30 PM	Dessert Break - Visit Exhibitors
1:30 PM	2:45 PM	**T11** An Introduction to TypeScript - *Jason Bock* / **T12** What's New for Developers in SQL Server 2016 - *Leonard Lobel*
3:00 PM	4:15 PM	**T16** Assembling the Web - A Tour of WebAssembly - *Jason Bock* / **T17** No Schema, No Problem! Introduction to Azure DocumentDB - *Leonard Lobel*
4:15 PM	5:30 PM	Welcome Reception
		Day 2: Wednesday, March 15, 2017
8:00 AM	9:15 AM	**W01** Angular 101: Part 1 - *Deborah Kurata* / **W02** Customizing Your UI for Mobile Devices: Techniques to Create a Great User Experience - *Laurent Bugnion*
9:30 AM	10:45 AM	**W06** Angular 101: Part 2 - *Deborah Kurata* / **W07** Windows for Makers: Raspberry Pi, Arduino & IoT - *Nick Landry*
11:00 AM	12:00 PM	**General Session:** .NET Everywhere - *James Montemagno, Principal Program*
12:00 PM	1:00 PM	Birds-of-a-Feather Lunch
1:00 PM	1:30 PM	Dessert Break - Visit Exhibitors - Exhibitor Raffle @ 1:15pm (Must be present to win)
1:30 PM	2:45 PM	**W11** User Authentication for ASP.NET Core MVC Applications - *Brock Allen* / **W12** Cloud Enable an Existing WPF LOB App - *Robert Green*
3:00 PM	4:15 PM	**W16** Securing Web APIs in ASP.NET Core - *Brock Allen* / **W17** Strike Up a Conversation with Cortana on Windows 10 - *Walt Ritscher*
4:30 PM	5:45 PM	**W21** ASP.NET Core 1.0 Tag Helpers - *Robert Boedigheimer* / **W22** Busy Developer's Guide to NoSQL - *Ted Neward*
7:00 PM	8:30 PM	Experience The LINQ Vortex & High Roller Event
		Day 3: Thursday, March 16, 2017
7:30 AM	8:00 AM	Registration - Coffee and Morning Pastries
8:00 AM	9:15 AM	**TH01** Debugging Your Website with Fiddler and Chrome Developer Tools - *Robert Boedigheimer* / **TH02** Busy .NET Developer's Guide to Native iOS - *Ted Neward*
9:30 AM	10:45 AM	**TH06** I Say A "Front-end Build Pipeline", You Say WAT!? - *Chris Klug* / **TH07** Building Cross-Platform Business Apps with CSLA .NET - *Rockford Lhotka*
11:00 AM	12:15 PM	**TH11** JavaScript Patterns for the C# Developer - *Ben Hoelting* / **TH12** Building Connected and Disconnected Mobile Apps - *James Montemagno*
12:15 PM	1:45 PM	Lunch
1:45 PM	3:00 PM	**TH16** Integrating AngularJS & ASP.NET MVC - *Miguel Castro* / **TH17** Native iOS and Android Development with C# and Xamarin - *James Montemagno*
3:15 PM	4:30 PM	**TH21** Increase Website Performance and Search with Lucene.Net Indexing - *Ben Hoelting* / **TH22** Building Cross-Platform C# Apps with a Shared UI Using Xamarin.Forms - *Nick Landry*
		Post-Conference Workshops: Friday, March 17, 2017 *(Separate entry fee required)*
7:30 AM	8:00 AM	Post-Conference Workshop Registration - Coffee and Morning Pastries
8:00 AM	5:00 PM	**F01** Workshop: Service Oriented Technologies - Designing, Developing, & Implementing WCF and the Web API - *Miguel Castro*

Speakers and sessions subject to change

Modern Apps LIVE!

MOBILE, CROSS-DEVICE & CLOUD DEVELOPMENT

Presented in partnership with Magenic

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

Visual Studio / .NET Framework	Web Client	Web Server	Windows Client	Modern Apps Live!

Full Day Hands-On Labs: Sunday, March 12, 2017 *(Separate entry fee required)*

Pre-Conference Workshop Registration - Coffee and Morning Pastries

HOL02 Full Day Hands-On Lab: AngularJS 2 - *Ted Neward*	**HOL03** Full Day Hands-On Lab: An Introduction to Building XAML Applications - *Billy Hollis*

Lunch @ Le Village Buffet, Paris Las Vegas

Hands-On Lab Continues	Hands-On Lab Continues

Pre-Conference Workshops: Monday, March 13, 2017 *(Separate entry fee required)*

Pre-Conference Workshop Registration - Coffee and Morning Pastries

M02 Workshop: Developer Dive into SQL Server 2016 - *Leonard Lobel*	**M03** Workshop: Distributed Cross-Platform Application Architecture - *Rockford Lhotka & Jason Bock*	**M04** Workshop: Building Modern Mobile Apps - *Brent Edwards & Kevin Ford*

Lunch @ Le Village Buffet, Paris Las Vegas

Workshop Continues	Workshop Continues	Workshop Continues

Dine-A-Round

Day 1: Tuesday, March 14, 2017

US Developer Division Team, Microsoft

T03 What's New in Visual Studio 2017 - *Robert Green*	**T04** Understanding the VR/AR Landscape - *Katherine Harris*	**T05** Modern App Development: Transform How You Build Web and Mobile Software - *Rockford Lhotka*
T08 Roll Your Own Dashboard in XAML - *Billy Hollis*	**T08** Mobile DevOps with the Microsoft Stack - *Abel Wang*	**T10** Manage Distributed Teams with Visual Studio Team Services and Git - *Brian Randell*

Lunch

Dessert Break - Visit Exhibitors

T13 A Developers Introduction to HoloLens - *Billy Hollis & Brian Randell*	**T14** Make PDF Work For You - *Aaron Schnarr*	**T15** Architecture: The Key to Modern App Success - *Brent Edwards*
T18 Essential C# 7.0 - *Mark Michaelis*	**T19** How to Scale .NET Apps with Distributed Caching - *Iqbal Khan*	**T20** Focus on the User Experience #FTW - *Jim Barrett*

Welcome Reception

Day 2: Wednesday, March 15, 2017

W03 What's New in Azure IaaS v2 - *Eric D. Boyd*	**W04** Tactical DevOps with VSTS - *Brian Randell*	**W05** DevOps, Continuous Integration, the Cloud, and Docker - *Dan Nordquist*
W08 Microservices with Azure Container Service & Service Fabric - *Vishwas Lele*	**W09** Use Visual Studio to Scale Agile in Your Enterprise - *Richard Hundhausen*	**W10** Mobile Panel - *James Montemagno, Ryan J. Salva, Kevin Ford, Rockford Lhotka*

Manager - Xamarin, Microsoft

Birds-of-a-Feather Lunch

Dessert Break - Visit Exhibitors - Exhibitor Raffle @ 1:15pm (Must be present to win)

W13 I'm Emotional - Using Microsoft Cognitive Services to Understand the World Around You - *Adam Tuliper*	**W14** Professional Scrum Development Using Visual Studio 2017 - *Richard Hundhausen*	**W15** C# Everywhere: How CSLA .NET Enables Amazing Cross-Platform Code Reuse - *Rockford Lhotka*
W18 Cloud Oriented Programming - *Vishwas Lele*	**W19** Introduction to Containers and Docker - *Marcel de Vries*	**W20** Coding for Quality and Maintainability - *Jason Bock*
W23 Practical Internet of Things for the Microsoft Developer - *Eric D. Boyd*	**W24** Using Docker on Windows in VSTS Build and Release Management - *Marcel de Vries*	**W25** Modern Mobile Development: Build a Single App for iOS & Android with Xamarin Forms - *Kevin Ford*

Experience The LINQ Vortex & High Roller Event

Day 3: Thursday, March 16, 2017

Registration - Coffee and Morning Pastries

TH03 Accelerate Your Mobile App Development with Azure App Services Mobile Apps - *Brian Noyes*	**TH04** Agile: You Keep Using That Word - *Philip Japikse*	**TH05** Modern Web Development: Building Server Side Using ASP.NET Core, MVC, Web API, and Azure - *Allen Conway*
H08 Connect All The Things with Azure Service Bus, Notification Hubs, Event Hubs, and IoT Hubs - *Brain Noyes*	**TH09** Visualizing the Backlog with User Story Mapping - *Philip Japikse*	**TH10** Modern Web Development: Building Client Side Using TypeScript and Angular - *Allen Conway*
TH13 Add A Conversational User Interface to Your App with the Microsoft Bot Framework - *Walt Ritscher*	**TH14** SOLID - The Five Commandments of Good Software - *Chris Klug*	**TH15** Security with Speed for Modern Developers - *Michael Lester*

Lunch

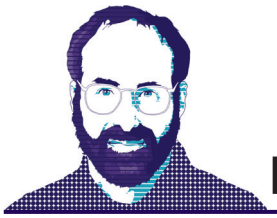
TH18 Introduction to R and Microsoft R Server - *James McCaffrey*	**TH19** Open Source Software for Microsoft Developers - *Rockford Lhotka*	**TH20** Universal Windows Development: UWP for PC, Tablet & Phone - *Nick Landry*
TH23 Introduction to Azure Machine Learning - *James McCaffrey*	**TH24** End-to-End Dependency Injection & Testable Code - *Miguel Castro*	**TH25** Using All That Data: Power BI to the Rescue - *Scott Diehl*

Post-Conference Workshops: Friday, March 17, 2017 *(Separate entry fee required)*

Post-Conference Workshop Registration - Coffee and Morning Pastries

F02 Workshop: Practical ASP.NET DevOps with VSTS or TFS - *Brian Randell*	**F03** Workshop: Creating Experiences for the HoloLens with Unity - *Nick Landry & Adam Tuliper*	**F04** Workshop: Modern App Deep Dive: Xamarin, Responsive Web, UWP, CSLA .NET - *Kevin Ford, Jason Bock, Brent Edwards, Allen Conway*

vslive.com/lasvegasmsdn



Stranger Things

In this column a year ago ("The Internet of Invisible Things," March 2016, msdn.com/magazine/mt683803), I mentioned the philosophy of Harry Shum, director of Microsoft Research: "Any sufficiently advanced technology is invisible." Today I want to analyze and develop another maxim of Harry's: "It's the user that's mobile, not the app."

We initially thought of a "mobile app" as an application that runs on a mobile device. The app is mobile because the user carries the device in his pocket, filling any and all gaps in his life: commuting to work ("Ring Around My Neck," February 2012, msdn.com/magazine/hh781031), waiting in line at the supermarket, sitting in the living room and so on.

But as mobile devices multiplied, users started owning more than one and switching between them, using different devices in different locations. Amazon was the first major company to recognize and develop this location neutrality with their family of Kindle reader devices and apps. Amazon remembers my behavior across my network of devices and apps, trying to anticipate my needs. It mostly succeeds.

For example, I read Kindle books on a Kindle Fire reader that I keep by my bedside. I also read them on my phone when I'm out and about, or on my other phone when my first one's broken or out of charge. Each app or device remembers what I was doing with it—that I'm working on the latest Jack Reacher thriller—and jumps to my most recently read page when I open them.

The new class of household voice-control speakers, such as Alexa and Google Home, are the logical evolution of this idea. Instead of having to carry a device from place to place, a connected device is waiting for you wherever you go. Some users love this arrangement, some hate it. I've seen the lovers place one in each room of the house because they're too lazy to walk from the den to the kitchen to order a pizza. (Is this user being mobile or *im*mobile?)

None of these electronic entities yet recognizes the different users in the household, but I'm sure it won't be long until that happens. For example:

Bobby (in his room): "Alexa, order a pizza."

Alexa: "Sorry, Bobby, your Mom said no more pizzas until you clean your room."

Mom (in the living room, hears Alexa on speaker there): "It's OK, Alexa, he did it. Sort of."

Alexa: "OK, Bobby, you got it. Want to try anchovies again?"

Bobby: "Yuck!"

Mom: "Alexa, just put them on my half."

Alexa: "OK, got it."

Working with these ubiquitous voice-recognizing household controllers gave me a weird sense of déjà vu, until I finally remembered where I'd seen them before—"The Addams Family" TV show (1964-1966). Alexa and Home are simply modern versions of Thing, the disembodied hand that appeared from a box in every room of the spooky Addams Family mansion. Like Alexa, Thing would deliver the mail in the format of the time (**Figure 1**). Thing also played music (**Figure 2**), auto-dialed phone contacts (**Figure 3**) and served as the world's first voice-controlled TV remote (**Figure 4**). Thing was truly the earliest app for the mobile user.

Figure 1 **Thing Delivering the Mail**

Figure 2 **Thing as Voice-Controlled Music Player**

Figure 3 **Thing as Contact Auto-Dialer**

Figure 4 **Thing as Voice-Controlled TV Remote**

That snotty ingrate Bobby didn't thank Alexa for his pizza, but I'm told that users often thank Alexa at the end of a transaction. I'm sure viewers will remember that Morticia Addams never failed to graciously say, "Thank you, Thing," for its assistance.

Each of today's digital assistants recognizes its own name in its wake-up sequence: "OK, Google," "Hey, Cortana," and so on. Amazon recently took a great step forward by releasing a new wake word for Alexa—you can now address her as "Computer," inspired by the Star Trek canon (engt.co/2jwsD93). I wonder which digital assistant will be first to respond to the name "Thing." ■

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.*

Looking for a powerful distributed cache?

Get the easiest, most powerful in-memory data grid designed to scale your .NET applications.

- Fast and linearly scalable
- Enterprise-grade availability
- Industry-leading ease of use
- Integrated in-memory computing

Go with the industry leader in distributed caching

Trusted by hundreds of enterprise customers for more than 12 years, ScaleOut's distributed caching technology delivers rock-solid performance, legendary ease of use, and world-class support. Unlike Redis, it offers a better migration path from AppFabric Caching. Here's why:

	ScaleOut	Redis
Source-code compatible AppFabric Caching APIs for seamless migration	✔	✖
Architected from the ground up for automatic scaling and high availability	✔	✖
Fully coherent data storage and access for mission-critical applications	✔	✖
Advanced .NET features: distributed LINQ query, C# MapReduce, and more	✔	✖

Replacing AppFabric Caching? Trouble switching to Redis?
Check out our AppFabric-compatible drop-in: **www.scaleoutsoftware.com/appfabric**

ScaleOut Software

Please visit **scaleoutsoftware.com** or call us at **+1.503.643.3422** to learn how ScaleOut's technology can help accelerate your business.

BIG DATA WHITE PAPER

STORE LIMITLESS AMOUNTS OF TIME SERIES DATA

WHAT YOU'LL LEARN:

- How time series data is processed at a rate of up to 100,000 records per second.
- The benefits of a real world Internet of Things scenario.
- How to access and display time series information in a visually appealing manner.
- The ease of scalability Syncfusion's Big Data Platform offers.

DOWNLOAD THIS WHITE PAPER TODAY!

www.syncfusion.com/MSDNbigdata

Syncfusion®