THE MICROSOFT JOURNAL FOR DEVELOPERS

NOVEMBER 2015 VOL 30 NO 12

msdn magazine

THE MICROSOFT JOURNAL FOR DEVELOPERS

NOVEMBER 2015 VOL 30 NO 12

msdn magazine

COLUMNS

Microsoft


BEST FILE APIs
Open Create Convert Print Save
files from your applications!
XLS
DOC
PDF
Contact Us:
US: +1 888 277 6734
EU: +44 141 416 1112
AU: +61 2 8003 5926
sales@aspose.com
SCAN FOR
20% SAVINGS!

msdn magazine

NOVEMBER 2015 VOLUME 30 NUMBER 12

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

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

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

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

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

Reaching the Staff

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

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

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

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

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

Microsoft

BPA WORLDWIDE

1906 100 2006 AMERICAN BUSINESS MEDIA

WPA Western Publications Association

Welcome, Essential .NET

If you were to glance at the Web metrics for published *MSDN Magazine* articles over the past five years, right away you'd notice that Mark Michaelis has his fingerprints all over the top of the board. His October 2014 feature, "The New and Improved C# 6.0" (msdn.com/magazine/dn802602), was the second-most visited article in the past five years, while his May 2014 feature, "A C# 6.0 Language Preview" (msdn.com/magazine/dn683793), wasn't far behind, ranking fourth in first-month page views out of more than 900 total published articles.

Success is hardly new to Michaelis, who's been a Microsoft MVP for going on 20 years now, and a Regional Director for the last eight. As chief executive officer and chief technical architect of consulting firm IntelliTect, he's spent the last nine years architecting and implementing advanced .NET-based solutions. And his close relationship with Microsoft has enabled him to emerge as a thought leader in the arena of .NET development. Not bad for a guy who majored in philosophy as an undergraduate (he went on to earn a master's degree in computer science).

All of which is a long-winded way to say Michaelis would make a terrific columnist at *MSDN Magazine*.

Starting this month, Michaelis' new Essential .NET column explores the broad development space around the Microsoft .NET Framework, starting with a look at exception handling in C# 6.0. The inaugural column will be the first of many, he says, to provide updated guidance and best practices for developers working with Microsoft's evolving flagship programming language.

"When .NET first came out, Brad Abrams and Krzysztof Cwalina spent countless hours educating the .NET community on the .NET Framework Design Guidelines. Since then, however, .NET and the .NET languages have changed and, along with that, the design guidelines have changed and improved, but without the same ambassadors," Michaelis says. "My focus on exception handling is just a first in a series of many articles to re-engage the developer community on writing maintainable, robust, performant and best practice-infused code."

What can you expect in the months to come? In December, look for an exploration of the design process around C# 7.0, followed later by a look at how the upcoming version will improve on C# 6.0. From there, expect Michaelis to dive into the deeper waters of the .NET Framework.

"My focus on exception handling is just a first in a series of many articles to re-engage the developer community on writing maintainable, robust, performant and best practice-infused code."

When I asked Michaelis why he thought his C#-themed features were so popular, he noted that at the time Microsoft's focus on Windows 8 had produced some "trepidation" among developers, who worried about the strategic commitment to the .NET Framework and C#. Michaelis says his articles were simply the right stuff at the right time: "My articles happened to come out at a time when there was a thirst by developers to learn the specifics of Microsoft's renewed commitment [to the language]," he says.

We're excited to have Michaelis on board, and look forward to seeing Essential .NET take up residence in our pages over the months and years to come. Do you have a topic or issue you would like to see Michaelis attack in an upcoming column? Let me know at mmeditor@microsoft.com.

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

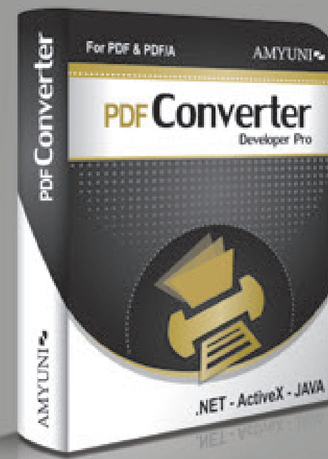
NEW v5.5

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

- 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

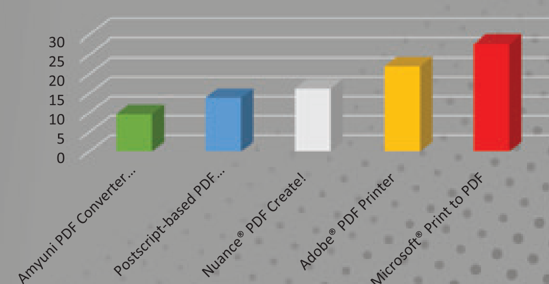
- All your PDF processing, conversion and editing in a single package
- Combines Amyuni PDF Converter and PDF Creator for easy licensing, integration and deployment.
- Includes our Microsoft WHQL certified PDF Converter printer driver
- Export PDF documents into other formats such as Jpeg, PNG, XAML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



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

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

All development tools available at
www.amyuni.com

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

Going Critical

A positive work environment is a great thing. It keeps people happy, which keeps them motivated and productive. When people feel good about their work, they go home feeling more fulfilled at the end of the day, and come back eager to do work that gains them more praise. This virtuous cycle can become a trap, however, when it enables a poorly designed feature into your software, or even worse, poorly designed software into your customer's hands.

To slip that trap, organizations must be able to engage in constructive criticism. If team members don't feel comfortable providing constructive feedback to others on the team, it won't be long before those criticisms come from angry customers, outraged over poor-quality products. The fact is, an issue not addressed in development will very likely impact product value in production and, ultimately, even damage your brand.

There is no sadder moment for a developer than to read a comment on Reddit complaining about something he observed before launch, but could do nothing about. This is something the game developer community has struggled with in recent years, with many large franchises pushing out high-profile titles that were, frankly, unfinished. Electronic Arts is a case in point. The difficult launch of its "Battlefield 4" game last year hurt the company's stock price and produced a shareholder lawsuit. Now it seems we're seeing more and more delayed game releases, as gamers refuse to line up to purchase new titles the way they had in the past.

There is no sadder moment for a developer than to read a comment on Reddit complaining about something he observed before launch, but could do nothing about.

Might early critical input during development have spared Electronic Arts and other game publishers a lot of trouble? Perhaps, but in a world ruled by ship deadlines and marketing target dates, the decision to delay a release to rework code is a tough one. It would help, perhaps, if there were a metric tracking the level of internal satisfaction for any piece of software in development. If that metric were to drop below a certain threshold, the product could be delayed until the issue is addressed, restoring internal satisfaction to an acceptable level.

Even if such a metric could be developed, the root issue remains: Modern software is often constrained by ship deadlines that create a hostile environment for critical input.

Even if such a metric could be developed, the root issue remains: Modern software is often constrained by ship deadlines that create a hostile environment for critical input. It's hard for a developer to constructively criticize another's work, when everyone knows there's no time or resources to do anything about it. It's also important to note that criticism can go too far. If a company's culture encourages employees to rip on each other's work, it can create a toxic environment that leaves people demoralized and impairs productivity.

Ultimately, developers can only control what they control. And a large part of that is their own reaction to received criticism. Too often people dismiss criticism. We all need to understand and accept that while we will naturally have a bias toward our own opinions, those offered by others are just as valid, and are often important in finding better solutions.

Like anything in life, the key is to find a happy balance. Organizations that sustain a culture of positive reinforcement, and support a process that allows teams to act on received criticism, are much more likely to produce high-quality software than those that shun these activities. And it all starts from a simple attitude adjustment. If people in your organization make the effort to support others' work, while valuing well-articulated, constructive criticism, it can transform your organization into a more productive and open workplace. ■

RYDER DONAHUE *is a software developer engineer at Microsoft. Originally from the Hawaiian Islands, he now resides in Redmond, Wash., with his fiancée and their cat, Marbles.*

Better Architecture with UX-Driven Design

The design and engineering of any software system begins with a well-known step: collecting requirements from users. This usually involves several meetings and interviews with customers and domain experts. Following the last meeting, nearly everyone involved in the project should believe that all details of the system have been ironed out and development can safely start. No one should doubt that the final product will be different from what was explained and understood. Customers should be happy and architects should believe they know exactly what to do.

However, in practice, experience shows that agreeing on abstract requirements doesn't guarantee successful implementation. When customers actually see the prototype, more often than not they just don't like it. No matter all the meetings and discussions, it seems that customers and developers form distinct ideas of the final product. Developers receive requirements and build the software around them. But the final product often misses the mark of what users want.

I think developers often tend to aim for functional completeness and don't focus enough on the business processes end users need for the system to perform. So while functional aspects of business processes are captured, the overall implementation is rarely as polished and smooth as expected. In this article, I'll present a design approach that can improve the chances for architects to build the right thing the first time. I call this approach UX-driven design, or UXDD for short.

The Apple That Fell on My Head

You may remember the apocryphal story of the apple that fell on Sir Isaac Newton's head, leading him to formulate the Universal Law of Gravitation. Recently, an apple fell on my head and the message I received was that paying due attention to users' expectations and real processes sometimes requires building different things. That is, it takes more than just a functional analysis.

A customer asked to build a quick-and-easy—so they said—Web application to support the competitive bracket for a tennis tournament. There was just one user story. The operator indicated the name of the player and the related position in the draw and the operator expected the system to expose some XML feed that reflected the current state of the draw. My developer's mindset immediately created the vision of a database table to hold the data. Next, I had the vision of some quick HTML form with two text boxes: one to indicate the player's name, and one for the position in the draw. Interestingly, when the discussion ended, the customer was certain he had a tool to enter players and positions and had some XML to support it. But when the developer—me—delivered just that and the customer tested it in the simulation of a live draw, it didn't work. The customer actually wanted something a lot more complex. A look at **Figure 1** shows different perspectives between developers and customers. The background screen is what the customer desires and the representation of the real process; the yellow screen with black outline in overlay is the low-cost solution, as quick as it is inappropriate.

At the end of the day, the UX is much more than just gestures and graphics, it's the experience that users go through when they interact with your software. To design an effective UX, as an architect and developer you should focus more on tasks and business processes than data models and storage. To understand tasks, you need to learn more about the domain, the users and how those users operate in that domain. UXDD addresses this, but it isn't simply a generic recommendation to use wireframes and mockups. I used that in a simple scenario and it didn't work because the customer—in his mind—thought the software was too simple and not worth the effort of fully engineering the real process. As an architect, I didn't get the right message from the customer about the importance of the task. Never choose a low-cost solution; choose an effective solution. I have to admit that the original solution I proposed—the low-cost solution—was just

Figure 1 **The Difference Between What's Wanted and What's Understood**

Figure 2 **Tools for Quick and Effective UI Prototyping**

Tool	URL
Axure	axure.com
Balsamiq	balsamiq.com
Indigo Studio	infragistics.com/products/indigo-studio
JustInMind	justinmind.com
UXPin	uxpin.com

impossible to use in a realistic situation. My fault was to entirely and blindly trust the customer's analysis and not learn more about the actual business processes.

UXDD is a set of architectural prescriptions that can minimize the risk of missing important business points that relate to tasks and the UI. Interestingly, UXDD can change some of the consolidated practices of today's development and software engineering.

Top-Down Design

From a functional perspective, you can successfully build a working system regardless of whether you start the design effort from the bottom (say, from the persistence model) or the top (say, from presentation layer and view model). From a UX perspective, you can only be successful if you start designing from presentation and view models and build everything else, including the back-end stack, from there.

Many of us learned professionally that once you had a persistence model made of entities and relationships, built from user requirements, you were pretty much done. That model would be the model of the entire system used throughout the entire stack and only occasionally partnered by a few UI-specific data-transfer objects. In this context, designing and building of the system take place in a bottom-up manner and the presentation layer inevitably reflects the persistence-centric vision of the data model. We often call this create, read, update, delete (CRUD) and we often reduce any system to CRUD with just a bit more sophisticated business logic. In doing so, we pay little attention to the presentation layer. Sometimes a UI too close to the data model is good for users; sometimes it is not. The latter scenario gives spark to additional negotiation once the first working prototype has been delivered. You first devise a system from the ground up to find out, at some point, that the outermost layer has to change to reflect different user processes. This is in no way different from trying to fit square pegs into round holes. Because of this, I see it as the biggest challenge of many software projects.

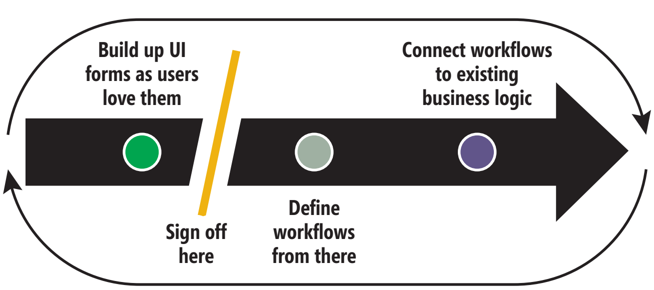


Figure 3 **The Three Steps of Architectural UX-Driven Design**

What can we do to improve the process? I believe the answer is to move back to a top-down design approach that starts from the presentation layer and makes any further decision and implementation detail descend from presentation needs. To make it effective, a sort of sprint zero, or waterfall preliminary step, is necessary to ensure that a deep understanding of the UX is captured before moving to build the back end of the system.

The UX-Driven Methodology

I learned from UX experts that requirements are better actively generated through evidence-based discussion than passively inferred via interviews. Sometimes architects and analysts tend to stay too passive during elicitation and this causes users to lower the priority of any features in order to have the software ready as soon as possible. Then they complain the system is barely usable when they finally get it. At the same time, staying too passive in the elicitation with the excuse that "this is what the customer wants" doesn't help make sense of the "thing" we're going to build. Today, we write software to mirror pieces of the real world rather than to model what we've been told the customer wants. Missing on structural aspects of the business is a deadly and costly sin.

> The UX is much more than just gestures and graphics.

It's a common practice to use wireframes to come to an agreement about the expected UI. Iteratively running wireframes by users to solicit feedback works only to a small extent. Wireframes are great, but of limited value without storyboards.

Very few tasks are entirely accomplished through a single screen that you can summarize effectively to a wireframe. Just looking into the wireframe of a screen may not be enough to spot possible bottlenecks of the process implementation. Concatenating screens in a storyboard is a much better idea. In this regard, the biggest challenge I see is finding the tools to build storyboards. These tools are a rapidly growing marketplace. **Figure 2** lists a few tools that can help you quickly prototype the presentation layer of applications in a way that gives users a concrete idea of the process being designed.

In addition, recent versions of Microsoft Visio and PowerPoint (especially in combination with Visual Studio Ultimate) also feature some prototyping capabilities. All tools listed in **Figure 2** offer a rich platform to create wireframes and in some cases offer the ability to create clickable mockups, and turn them into functional prototypes.

The most advanced of these tools provide early feedback and, more important, let you involve customers earlier in the design process and before writing a single line of code. If you find you missed important presentation points when half the back end is done, you either throw it away or adjust things.

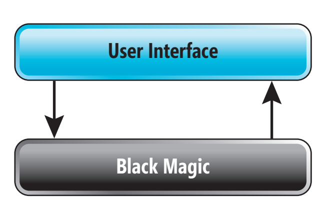


Figure 4 **Essence of Software from the User's Perspective**

At the same time, simply outsourcing the presentation layer to a team of UX experts isn't enough. The presentation layer today is the most important part of a system and must result from the combined effort of solution architects, UX architects and customers. This must be the first step and ideally you move on only when the customer signs off on the presentation. In terms of methodology, it's acceptable to take a waterfall slant and complete the entire presentation design before coding, and be more agile and add the presentation analysis as a step in the sprint, as shown in **Figure 3**.

Designing the Rest of the Solution

Once you've finalized the presentation layer for the entire solution or just the current sprint, you have a collection of screens—for example, forms—with a well-defined data flow (it's clear what comes in and out of each form). Architecturally speaking, this means you know the input model of each action and the view model necessary to fill out the form or generate the expected response. The presentation layer connects to the back end via an intermediate service layer that conceptually matches the Service Layer pattern as defined by Martin Fowler (bit.ly/1JnFk8i), as well as the application layer in the layered architecture of Domain-Driven Design. It is the logical segment of the system where you implement the use cases and any orchestration of tasks they require. The application layer is the topmost part of the back end and dialogs directly with the presentation layer. The application layer is made of direct endpoints for each of the actions that can be triggered by the presentation. These endpoints receive and return just the input and view models resulting from wireframes.

Is this approach really effective? You bet. If your wireframe analysis is thorough and correct, you're implementing just the processes customers want, and it's right the first time. You cut down significantly the chances of renegotiating changes after you deploy the first release or demo. This saves time and, subsequently, money. As shown in **Figure 4**, developers indicate this as emblematic of how users look at software. And UXDD leads to designing software in that way.

Wrapping Up

Compared to how we design software today—from the data model up—UXDD assigns more importance to tasks and presentation than data models. Not that data modeling and persistence are unimportant, but their roles are functional to tasks rather than the other way around. Like it or not, this is closer to what the real world demands today. UXDD is about methodology rather than technology or patterns. UXDD doesn't deny nor mandate any technology or pattern, though it goes very well together with CQRS and Event Sourcing. If you're not satisfied with the actual process of building applications, use the UXDD approach as a form of lateral thinking. ■

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

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

msdn magazine

STATEMENT OF OWNERSHIP, MANAGEMENT AND CIRCULATION

1. Publication Title: *MSDN Magazine*
2. Publication Number: 1528-4859
3. Filing Date: 9/30/15
4. Frequency of Issue: Monthly with a special issue in October.
5. Number of Issues Published Annually: 13
6. Annual Subscription Price: US $35, International $60
7. Complete Mailing Address of Known Office of Publication: 9201 Oakdale Ave., Ste. 101, Chatsworth, CA 91311
8. Complete Mailing Address of the Headquarters of General Business Offices of the Publisher: Same as above.
9. Full Name and Complete Mailing Address of Publisher, Editor, and Managing Editor:
   Henry Allain, President, 4 Venture, Suite 150, Irvine, CA 92618
   Michael Desmond, Editor-in-Chief, 8609 Westwood Center Dr., Suite 500, Vienna, VA 22182
   Wendy Hernandez, Group Managing Editor, 4 Venture, Ste. 150, Irvine, CA 92618
10. Owner(s): 1105 Media, Inc. dba: 101 Communications LLC, 9201 Oakdale Ave, Ste. 101, Chatsworth, CA 91311. Listing of shareholders in 1105 Media, Inc.
11. Known Bondholders, Mortgagees, and Other Security Holders Owning or Holding 1 Percent or more of the Total Amount of Bonds, Mortgages or Other Securities:
    Nautic Partners V, L.P., 50 Kennedy Plaza, 12th Flr., Providence, RI 02903
    Kennedy Plaza Partners III, LLC, 50 Kennedy Plaza, 12th Flr., Providence, RI 02903
    Alta Communications IX, L.P., 1000 Winter Street, South Entrance, Suite 3500, Waltham, MA 02451
    Alta Communications IX, B-L.P., 1000 Winter Street, South Entrance, Suite 3500, Waltham, MA 02451
    Alta Communications IX, Associates LLC, 1000 Winter Street, South Entrance, Suite 3500, Waltham, MA 02451
12. The tax status has not changed during the preceding 12 months.
13. Publication Title: *MSDN Magazine*
14. Issue date for Circulation Data Below: September 2015
15. Extent & Nature of Circulation:

	Average No. Copies Each Month During Preceding 12 Months	No. Copies of Single Issue Published Nearest to Filing Date
a. Total Number of Copies (Net Press Run)	84,290	81,823
b. Legitimate Paid/and or Requested Distribution		
1. Outside County Paid/Requested Mail Subscriptions Stated on PS Form 3541	67,418	65,476
2. In-County Paid/Requested Mail Subscriptions Stated on PS Form 3541	0	0
3. Sales Through Dealers and Carriers, Street Vendors, Counter Sales, and Other Paid or Requested Distribution Outside USPS®	2,303	2,687
4. Requested Copies Distributed by Other Mail Classes Through the USPS®	0	0
c. Total Paid and/or Requested Circulation	69,721	68,163
Nonrequested Distribution		
1. Outside County Nonrequested Copies Stated on PS Form 3541	11,247	12,497
2. In-County Nonrequested Copies Distribution Stated on PS Form 3541	0	0
3. Nonrequested Copies Distribution Through the USPS by Other Classes of Mail	0	0
4. Nonrequested Copies Distributed Outside the Mail	3,169	1,015
e. Total Nonrequested Distribution	14,416	13,512
f. Total Distribution	84,137	81,675
g. Copies not Distributed	153	148
h. Total	84,290	81,823
i. Percent paid and/or Requested Circulation	82.87%	83.46%

16. Electronic Copy Circulation
    a. Requested and Paid Electronic Copies
    b. Total Requested and Paid Print Copies (Line 15c) + Requested/Paid Electronic Copies
    c. Total Requested Copy Distribution (Line15f) + Requested/Paid Electronic Copies (Line 16a)
    d. Percent Paid and/or Requested Circulation (Both print & Electronic Copies) (16b divided by 16c x 100)
    ☑ **I certify that 50% of all my distributed copies (electronic and paid print are legitimate request or paid copies.**
17. Publication of Statement of Ownership for a Requester Publication is required and will be printed in the November 2015 issue of this publication.
18. I certify that all information furnished on this form is true and complete:
    David Seymour, Director, Print and Online Production

JULIE LERMAN

Aurelia Meets DocumentDB: A Matchmaker's Journey

In the past months you've seen, via this column, my explorations into worlds unknown. My September column delved into data binding in an upstart JavaScript client framework called Aurelia, which communicated with a back-end ASP.NET 5 Web API to retrieve and store data. That Web API was using SQL Server and Entity Framework behind the scenes. My June column explored the new Microsoft Azure NoSQL database service, DocumentDB. In that column I also built an ASP.NET MVC 5 Web API, but that Web API leveraged the DocumentDB .NET client library to communicate with a DocumentDB database.

With the wheels greased for Aurelia and DocumentDB, I wanted to put those two together. I headed down some bumpy roads and hit some walls, but eventually wound up on the correct path to allow Aurelia to communicate with DocumentDB. In this column, I'll share some of the lessons I learned along the way and an overview of my final solution. In an upcoming column, I'll provide more detail about that solution.

While you'll certainly learn a lot from the solution I eventually came up with, I think the steps I took along the way, even those that led to failure, are equally educational.

While you'll certainly learn a lot from the solution I eventually came up with, I think the steps I took along the way, even those that led to failure, are equally educational.

I'll begin by sharing my plan—which originated in a vacuum of inexperience with JavaScript and its many APIs, tools and patterns—then reveal what seemed at the time like great ways to achieve my goal. If I instead went straight to the correct path, chances are you'd second guess my choice and try to find a better way, only to arrive at the same dead-ends I've already visited. Then, of course, I'll share with you the true path, although it takes more than a single installment of this column to tell the whole story.

Code download available at msdn.com/magazine/msdnmag1115.

Best Laid Plans: "How Hard Can It Be?"

Azure DocumentDB is a service, and raw interaction with it is through either a SQL syntax or RESTful HTTP calls. In the previous column, rather than work at that low level, I took advantage of one of the many APIs that Microsoft has created for DocumentDB—the .NET API. This allowed me to use LINQ to express and execute queries against my database, such as:

```
return Client.CreateDocumentQuery(Collection.DocumentsLink)
             .Where(d => d.Id == id)
             .AsEnumerable()
             .FirstOrDefault();
```

I wrote a Web API to do this work for me.

In the demo for my column on Aurelia data binding, I was able to easily make HTTP calls to a different Web API, one that used Entity Framework to hit a SQL Server database. I could've just replaced that Web API with the one I had built to talk to DocumentDB. And in that scenario, my job would be done and this column would be the shortest one by me you've ever read. How boring.

Instead, I thought it would be more interesting to use the direct REST interaction that DocumentDB provides. It seemed simple enough. The DocumentDB documentation shows quite a few examples of HTTP requests made to DocumentDB, which I thought I'd be able to leverage. **Figure 1** shows one example.

But it's not quite so easy. To start, the master key, which you see in the authorization, isn't something you want to provide inside a client application. Furthermore, it isn't truly the master key that comes from DocumentDB settings in the Azure portal; it's a hash of the key plus additional information. The "Access Control on DocumentDB Resources" article at bit.ly/109dBfP describes how to

Figure 1 An Example HTTP Request to DocumentDB

```
POST https://contosomarketing.documents.azure.com/dbs/XP0mAA==/colls/
XP0mAJ3H-AA=/docs HTTP/1.1
x-ms-DocumentDB-isquery: True
x-ms-date: Mon, 18 Apr 2015 13:05:49 GMT
authorization: type%3dmaster%26ver%3d1.0%26sig[A HASH GOES HERE]
x-ms-version: 2015-04-08
Accept: application/json
Content-Type: application/query+json
Host: contosomarketing.documents.azure.com
Content-Length: 50
{
  query: "SELECT * FROM root WHERE (root.Author.id = 'Don')",
  parameters: []
}
```

Figure 2 **The showTasks Command in the tasklist.js Controller Class**

```
showTasks: function (request, response) {
  var self = this;

  var querySpec = {
    query: 'SELECT * FROM root r WHERE r.completed=@completed',
    parameters: [{
      name: '@completed',
      value: false
    }]
  };

  self.taskDao.find(querySpec, function (err, items) {
    if (err) {
      throw (err);
    }

    response.render('index', {
      title: 'My ToDo List ',
      tasks: items
    });
  });
},
```

construct this string, but even this didn't enable me to test these API calls in Fiddler.

But the bigger problem is that I wouldn't be doing this in the client application, anyway. The recommended architecture according to "Securing Access to DocumentDB Data" (bit.ly/1N2ZiuF) is to create middleware that has safe access to your master key and is able to generate and return resource keys for a client-side application to use.

So I did that. I built an ASP.NET Web API using the .NET client for DocumentDB to return a properly composed resource key on demand. This entailed all of the same setup code that was part of my earlier DocumentDB column, where I defined my Azure account, the database client, the database within that client and the collections within that database. On top of that I had an 85-line controller that connected to the database; checked for a user; checked for, cleared out and recreated permissions for that user for the action I was about to perform; then generated, hashed and returned a resource token to the requesting client. That was a pretty complicated bit of code to comprehend and set up.

With that service in place, my next step was to have my Aurelia app call that service to retrieve a token for a given operation and then reuse that token in that operation. This isn't much different in the long run than security operations we use, for example, for Windows Communication Foundation (WCF) services. But it made for a very, very chatty solution. And, in the end, I still was unable to construct a proper request with my generated resource tokens on the client side (JavaScript) because there were more complexities required by DocumentDB. DocumentDB refused to authorize my requests to retrieve data. So I decided that making my own RESTful calls directly from a client app via HTTP to my DocumentDB was a path that wasn't going to lead to success. I do, however, anticipate that as DocumentDB evolves, so will our options for connecting to it, and I plan to revisit this idea in the future.

Still, all was not lost. DocumentDB also has a JavaScript SDK. This SDK would know how to construct the RESTful calls to DocumentDB on my behalf even if I was using its higher-level methods. I brought the SDK into my client solution with the understanding that I could let it construct the request for me using resource tokens requested from my ResourceTokenGenerator Web API. This felt like the right path, finally, but in the end I hit yet another wall: There was no way to enable cross-origin resource sharing (CORS), which meant that calls from my client-side app on one domain to my service on another would not be allowed.

At this point, having exhausted my resources and my curiosity about making RESTful calls without a wrapper to do it for me, and still not wanting to just flip to my existing Web API to get my Aurelia app to talk to DocumentDB, I headed down another road.

Success: DocumentDB, Express, Aurelia and Node.js

In addition to the. NET and JavaScript client SDKs, DocumentDB also provides a Node.js SDK (bit.ly/1LifOa1). This lets you use Node.js, a JavaScript implementation that works on the server side—much like ASP.NET code-behind logic—to easily access DocumentDB. All of the hard parts of configuration, authentication and building the RESTful API calls are wrapped into methods of the SDK. So I decided this was the path I'd have to follow to let my Aurelia application talk to DocumentDB. It meant a lot of new learning hurdles for me. I'd never touched Node.js and am famously a JavaScript noob; moreover, it involved an additional API, Express, which wraps up a bunch of core functionality to make it easier to use Node.js. But that's not all. For my first dive into Aurelia, I had to get used to working at the command line and using SublimeText, a text editor that's much savvier for Web development than Notepad. Because most of the action in that earlier app was on the client side, I was able to debug right in the browser. But now I was debugging Node.js code, which is on the server. It turned out that the new addition to the Visual Studio family, Visual Studio Code, is a great tool for that.

> Azure DocumentDB is a service, and raw interaction with it is through either a SQL syntax or RESTful HTTP calls.

Thankfully, I was able to benefit from two key samples. On the DocumentDB side, there's a walk-through for building a small Web application with Node.js and DocumentDB (bit.ly/1FljQs6). On the Aurelia side, there's a repository on GitHub that sets up a skeleton Aurelia app with Node.js server-side logic already integrated (bit.ly/1XkMuEX).

In order to implement my solution, it was critical to have a good understanding of the underlying mechanics of the sample that uses the Node.js SDK. I'll spend the rest of this column exposing details beyond what the high-level walk-through provides. This will set you up for the reveal of how I wired up the API in my final solution.

The DocumentDB Node.js walk-through provides back-end logic in Node.js that leverages the DocumentDB Node.js SDK to

communicate with DocumentDB. The first chunk of this logic is a pair of generic utilities that takes care of creating an instance of the database connection, creating the database first if necessary, and creating an instance of the specific collection within the database with which to work. These utilities can be reused in other applications to communicate with any DocumentDB because DocumentDB allows you to pass in authentication information and other information to specify the database and collections with which you're working.

This utilities class starts by making a reference to the DocumentDB SDK for Node.js that has already been installed into the solution:

```
var DocumentDBClient = require('DocumentDB').DocumentClient;
```

It then sets up the methods that take in the relevant connection information as parameters. Here, for example, is the class declaration with the beginning of the getOrCreateDatabase method, which first defines a query to get the database from Azure, then uses the queryDatabases method of the SDK DocumentClient class to execute that query. If the results are empty, then another call (not shown) will create the database for you. The database instance is returned from the method. You can see the full listing of the DocDBUtils class in the referenced article (bit.ly/1FljQs6):

```
var DocDBUtils = {
  getOrCreateDatabase: function (client, databaseId, callback) {
    var querySpec = {
      query: 'SELECT * FROM root r WHERE r.id=@id',
      parameters: [{
        name: '@id',
        value: databaseId
      }]
    };

    client.queryDatabases(querySpec).toArray(function (err, results) {
    // Additional logic to specify callbacks and more
```

The second chunk of logic, in a file called tasklist.js, is akin to a controller. It offers methods to leverage the database and collection instances provided by the DocDBUtils class so you can interact with the data. This controller is designed specifically for the sample that stores and retrieves ToDo items—Tasks. The Task object is encapsulated in a class called TaskDao, and you'll see references to an instance of TaskDao in the controller class. The controller has methods to retrieve Tasks, as well as to add new ones, and to update and delete. The class begins with references to the DocumentDB SDK, as well as the utility class I've just described:

```
var DocumentDBClient = require('DocumentDB').DocumentClient;
var docdbUtils = require('./docdbUtils');
```

Tasklist.js includes functions such as showTasks and addTask. These functions follow Node.js conventions by taking as parameters a request object and a response object that allow Node.js to either pass the request from the browser along to another process or inject whatever it wants into the response, which will get passed back to the browser. **Figure 2** displays the showTasks function.

Keep in mind that there's one additional library—Express—being used in this sample. Express wraps Node.js features into higher-level methods. Notice that the showTasks function uses an Express render method of the response object to render the index view (such as index.html), passing the items retrieved from DocumentDB into the tasks property that will be available to use in the Index.html file.

The controller (the TaskList class) is the first entry point into the server-side Node.js logic as it responds to a Web site's routing. The logic in the controller methods uses the taskDAO object to trigger queries and updates as shown by the call to self.taskDao.find in the showTasks function. taskDAO has an init function that uses the DocDbUtils to set up the database and collection to be used. With those in hand, it can then use the DocumentDB SDK directly to define and execute the queries and updates in its find, getitem and updateItem functions, as **Figure 3** shows.

With the back-end logic set up in Node.js, the next step is to build the front end. The walk-through on the DocumentDB site uses a view-generation framework called Jade. With HTML files for views and routing set up using the Jade API, the UI is able to respond to user navigation requests by calling into the taskList controller on the server, where I can safely store my DocumentDB keys to authorize my interactions with the data.

Next Steps: Hooking Up Aurelia to the Node.js Back End

Remember, though, that my goal was to use Aurelia as the client framework, not Jade. What came next was taking the lessons about using the DocumentDB Node.js SDK and applying them to the Node.js-enabled skeleton application provided by the Aurelia-node sample on GitHub. Aurelia routing works a bit differently from Jade routing, however, and it wasn't just a matter of "clicking" these two puzzle pieces together. My inexperience with Node.js and Express, together with my general "knows-enough-to-be-dangerous" JavaScript skills made the challenge much greater than it needed to be. But I did eventually work it all out with help from a number of the Aurelia core team members.

In my next column I'll walk you through the critical connectors between the controller and the Aurelia routing and show how using the server-side Node.js solution to interact with my DocumentDB compared to the simplicity of making direct HTTP calls from Aurelia to a Web API. ■

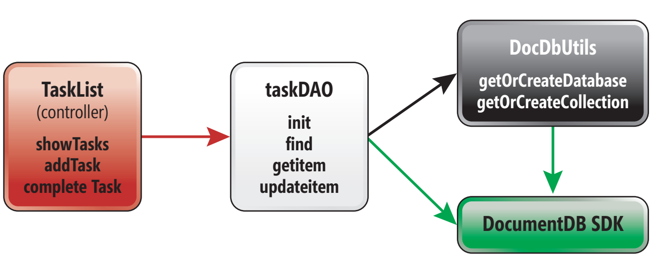


Figure 3 **Workflow Dependencies of the DocumentDb Node.js Sample Application Classes and SDK**

Julie Lerman *is a Microsoft MVP, .NET mentor and consultant who lives in the hills of Vermont. You can find her presenting on data access and other .NET topics at user groups and conferences around the world. She blogs at thedatafarm.com/blog and is the author of "Programming Entity Framework" (2010), as well as a Code First edition (2011) and a DbContext edition (2012), all from O'Reilly Media. Follow her on Twitter: @julielerman and see her Pluralsight courses at juliel.me/PS-Videos.*

Thanks to the following technical experts for reviewing this article: *Ryan CrawCour and Patrick Walters*

Accelerate File Operations with the Search Indexer

Adam Wilson

The search indexer has been a part of Windows for many releases now, powering everything from the library views in File Explorer to the IE address bar, as well as providing search functionality for the Start menu and Outlook. With Windows 10, the power of the indexer has gone from being limited to desktop machines, to being available to all Universal Windows Platform (UWP) apps. While this also lets Cortana run better searches, the most exciting part of this advancement is that it greatly improves how apps interact with the file system.

The indexer lets apps do more interesting operations such as sorting and grouping files and tracking how the file system is changing. Most of the indexer APIs are available to UWP apps through the Windows.Storage and Windows.Storage.Search namespaces. Apps are already using the indexer to enable great experiences for their users. In this article, I'll walk through how you can use the indexer to track changes on the file system, render views quickly and offer some basic tips on how to improve an app's queries.

This article discusses:

- With Windows 10, how search indexer now works with all Universal Windows Platform apps
- How the Windows 10 search indexer improves how apps interact with the file system
- How to use the indexer to track changes on the file system, render views quickly and tips to improve an app's queries

Technologies discussed:

Windows 10, Search Indexer, Windows.Storage, Universal Windows Platform, Cortana

Accessing Files and Metadata Quickly

Most user devices contain hundreds or thousands of media files that include a user's most cherished pictures and favorite songs. Apps that can quickly iterate through the files on the device and offer stimulating interactions with the files are among the most-loved apps on any platform. The UWP provides a series of classes that can be used to access files on any device, regardless of the form factor.

The Windows.Storage namespace includes the basic classes for accessing files and folders, as well as the base operations that most apps do with them. But if your app needs to access a lot of files or metadata then these classes won't provide the performance characteristics that users demand.

For example, calling StorageFolder.GetFilesAsync is a recipe for disaster if you don't control the folder you're enumerating. Users can put billions of files in a single directory, but attempting to create StorageFile objects for each of them will cause an app to run out of memory very quickly. Even in less extreme cases the call will still return very slowly because the system has to create thousands of file handles and marshal them back into the app container. To help apps avoid this pitfall, the system provides the StorageFileQueryResults and StorageFolderQueryResults classes.

StorageFileQueryResults is the go-to class whenever writing an app to handle more than a trivial number of files. Not only does it provide a neat way to enumerate and modify the results of a complex search query, but because the API treats an enumeration request as a query for "*," it also works for more mundane cases.

Using the indexer where available is the first step to speeding up your app. Now, coming from the indexer program manager, that sounds like a self-serving plea to keep me employed, but there's a

Figure 1 GetFilesAsync

```
uint index = 0, stepSize = 10;
IReadOnlyList<StorageFile> files = await queryResult.GetFilesAsync(index, stepSize);
index += 10;
while (files.Count != 0)
{
  var fileTask = queryResult.GetFilesAsync(index, stepSize).AsTask();
  foreach (StorageFile file in files)
  {
    // Do the background processing here
  }
  files = await fileTask;
  index += 10;
}
```

logical reason I say that. The StorageFile and StorageFolder objects were designed with the indexer in mind. The properties cached in the object can be retrieved quickly from the indexer. If you aren't using the indexer, the system has to look up values from the disk and registry, which is I/O-intensive and causes performance issues for both the app and the system.

To make sure the indexer is going to be used, create a QueryOptions object and set QueryOptions.IndexerOption property to either only use the indexer:

```
QueryOptions options = new QueryOptions();
options.IndexerOption = IndexerOption.OnlyUseIndexer;
```

or, use the indexer when it's available:

```
options.IndexerOption = IndexerOption.UseIndexerWhenAvailable;
```

The recommended usage is for cases where a slow file operation won't lock up your app or cripple the UX to use IndexerOption.UseIndexerWhenAvailable. This will attempt to use the indexer to enumerate files, but fall back to the much slower disk operations, if needed. IndexerOption.OnlyUseIndexer is best used when returning no results is better than waiting for a slow file operation. The system will return zero results if the indexer is disabled, but will return quickly, still letting apps be reactive to the user.

There are times that creating a QueryOptions object seems a little excessive for just a quick enumeration and, in those cases, it might make sense to not worry if the indexer is present. For cases where you control the contents of the folder, calling StorageFolder.GetItemsAsync makes sense. It's an easy line of code to write and any perf issues will be hidden in cases where there are only a few files in the directory.

> Another way to speed up file enumeration is to not create unnecessary StorageFile or StorageFolder objects.

Another way to speed up file enumeration is to not create unnecessary StorageFile or StorageFolder objects. Even when using the indexer, opening a StorageFile requires the system to create a file handle, gather some property data, and marshal it into the app's process. This IPC comes with inherent delays, which can be avoided in many cases by simply not creating the objects in the first place.

An important thing to note, a StorageFileQueryResult object backed by the indexer doesn't create any StorageFiles internally. They're created on demand when requested through GetFilesAsync. Until that time, the system only keeps a list of the files in memory, which is comparatively lightweight.

The recommended way to enumerate a large number of files is to use the batching functionality on GetFilesAsync to page in groups of files as they're needed. This way, your app can do background processing on the files while it's waiting for the next set to be created. The code in **Figure 1** shows how it's done in a simple example.

This is the same coding pattern that has been used by a number of apps already on Windows. By varying the step size they're able to pull out the right number of items to have a responsive first view in the app, while quickly prepping the rest of the files in the background.

> The StorageFile and StorageFolder objects were designed with the indexer in mind.

Property prefetching is another easy way to speed up your app. Property prefetching lets your app notify the system that it's interested in a given set of file properties. The system will grab those properties from the indexer while it's enumerating a set of files and cache them in the StorageFile object. This provides an easy performance boost versus just gathering the properties piecemeal when the files are returned.

Set the property prefetch values in the QueryOptions object. A few common scenarios are supported by using the PropertyPrefetchOptions, but apps are also able to customize the properties requested to any values supported by Windows. The code to do this is simple:

```
QueryOptions options = new QueryOptions();
options.SetPropertyPrefetch(PropertyPrefetchOptions.ImageProperties,
  new String[] { });
```

In this case, the app uses the image properties and doesn't need any other properties. When the system is enumerating the results of the query, it will cache the image properties in memory so they're quickly available later.

One last note is that the property has to be stored in the index for the prefetching to offer a performance gain; otherwise, the system will still have to access the file to find the value, which is comparatively very slow. The Microsoft Windows Dev Center page for the property system (bit.ly/1LuovhT) has all the information about the properties available on Windows indexer. Just look for isColumn = true in the property description, which will indicate the property is available to be prefetched.

Bringing all of these improvements together lets your code run much faster. For a simple example, I wrote an app that retrieves all the pictures on my computer along with their vertical height. This is the first step that a photo-viewing app would have to take in order to display the user's photo collection.

I did three runs to try out different styles of file enumeration and to show the differences between them. The first test used naïve code with the indexer enabled as shown in **Figure 2**. The second test used the code shown in **Figure 3**, which does property prefetching and pages in files. And the third is using property prefetching and paging files in, but with the indexer disabled. This code is the same as in **Figure 3**, but with one line changed as noted in the comments.

Taking a look at the results with and without the prefetch, the performance difference is really clear, as shown in **Figure 4**.

There's a chance to almost double the performance of naïve code by applying the simple optimizations outlined here. The patterns are battle-tested, as well. Before releasing any version of Windows we work with the app teams to make sure Photos, Groove Music and others are working as well as possible. That is where these patterns came from; they were cribbed directly from the code of the first UWP apps on the UWP and can be applied directly to your apps.

There are two methods for change tracking depending on if your app is in the background or foreground.

Tracking Changes in the File System

As shown here, enumerating all the files in a location is a resource-intensive process. Most of the time, your users aren't even going to be interested in older files. They want the picture that they just took, the song that was just downloaded, or the most recently edited document. To help bring the most recent files to the top, your app can track changes to the file system and find the most recently created or modified files easily.

There are two methods for change tracking depending on if your app is in the background or foreground. When an app is in the foreground, it can use the ContentsChanged event from a StorageFileQueryResult object to be notified of changes under a given query. When an app is in the background, it can register for the StorageLibraryContentChangedTrigger in order to be notified when something changed. Both of these are poke notifications to let an app know that something changed, but don't include information about the files that have changed.

To find which files have been modified or created recently, the system provides the System.Search.GatherTime property. The property is set for all files in an indexed location and tracks the last time that the indexer noticed a modification to the file. This property will be constantly updated based off the system clock, though, so the usual disclaimers involving time zone switches, daylight saving and users manually changing the system time still apply to trusting this value in your app.

Registering for a change tracking event in the foreground is easy. Once you've created a StorageFileQueryResult object covering the scope that your app is interested in, simply register for the ContentsChanged event, as shown here:

```
StorageFileQueryResult resultSet = photos.CreateFileQueryWithOptions(option);
resultSet.ContentsChanged += resultSet_ContentsChanged;
```

The event is going to be fired any time that something in the result set changes. The app can then find the file or files that changed recently.

Tracking changes from the background is slightly more involved. Apps are able to register to be notified when a file changes under a library on the device. There's no support for more complex queries or scopes, which means that apps are responsible for doing a bit of work to make sure that the change is something in which they're really interested.

As an interesting aside, the reason apps can only register for the library change notifications and not based on file type is due to how the indexer is designed. Filtering queries based on the location of a file on disk is much faster than querying for a match based on the file type; and, it dragged down the performance of devices in our initial tests. I will cover more performance tips later, but this is an

Figure 2 **Naïve Code Enumerating a Library**

```
StorageFolder folder = KnownFolders.PicturesLibrary;
QueryOptions options = new QueryOptions(
  CommonFileQuery.OrderByDate, new String[] { ".jpg", ".jpeg", ".png" });
options.IndexerOption = IndexerOption.OnlyUseIndexer;
StorageFileQueryResult queryResult = folder.
CreateFileQueryWithOptions(options);

Stopwatch watch = Stopwatch.StartNew();
IReadOnlyList<StorageFile> files = await queryResult.GetFilesAsync();

foreach (StorageFile file in files)
{
  IDictionary<string, object> size =
    await file.Properties.RetrievePropertiesAsync(
    new String[] { "System.Image.VerticalSize" });
  var sizeVal = size["System.Image.VerticalSize"];
}
watch.Stop();
Debug.WriteLine("Time to run the slow way: " + watch.ElapsedMilliseconds + " ms");
```

Figure 3 **Optimized Code for Enumerating a Library**

```
StorageFolder folder = KnownFolders.PicturesLibrary;

QueryOptions options = new QueryOptions(
  CommonFileQuery.OrderByDate, new String[] { ".jpg", ".jpeg", ".png" });
// Change to DoNotUseIndexer for trial 3
options.IndexerOption = IndexerOption.OnlyUseIndexer;

options.SetPropertyPrefetch(PropertyPrefetchOptions.None, new String[] {
"System.Image.VerticalSize" });
StorageFileQueryResult queryResult = folder.CreateFileQueryWithOptions(options);

Stopwatch watch = Stopwatch.StartNew();
uint index = 0, stepSize = 10;
IReadOnlyList<StorageFile> files = await queryResult.GetFilesAsync(index, stepSize);
index += 10;

// Note that I'm paging in the files as described
while (files.Count != 0)
{
  var fileTask = queryResult.GetFilesAsync(index, stepSize).AsTask();
  foreach (StorageFile file in files)
  {
// Put the value into memory to make sure that the system really fetches the property
    IDictionary<string,object> size =
      await file.Properties.RetrievePropertiesAsync(
      new String[] { "System.Image.VerticalSize" });
    var sizeVal = size["System.Image.VerticalSize"];
  }
  files = await fileTask;
  index += 10;
}
watch.Stop();
Debug.WriteLine("Time to run: " + watch.ElapsedMilliseconds + " ms");
```


important one to remember: Filtering query results by file location is extremely fast compared to other types of filtering operations.

I've outlined the steps to register for a background task with code samples in a blog post (bit.ly/1iPUVIo), but let's walk through a couple of the more interesting steps here. The first thing an app must do is create the background trigger:

```
StorageLibrary library =
  await StorageLibrary.GetLibraryAsync(KnownLibraryId.Pictures);
StorageLibraryContentChangedTrigger trigger =
  StorageLibraryContentChangedTrigger.Create(library);
```

The trigger can also be created from a collection of libraries if the app is going to be interested in tracking multiple locations. In this case, the app is only going to be looking at the pictures library, which is one of the most common scenarios for apps. You need to be sure the app has the correct capabilities to be able to access the library it's trying to change track; otherwise, the system will return an access denied exception when the app tries to create the StorageLibrary object.

On Windows mobile devices, this is especially powerful as the system guarantees new pictures from the device are going to be written under the pictures library location. This is done no matter what the user chooses in the settings page by changing which folders are included as part of the library.

The app must register the background task using the BackgroundExecutionManager and have a background task built into the app. The background task could be activated while the app is in the foreground, so any code must be aware of potential race conditions on file or registry access.

Once the registration is done, your app is going to be called every time there's a change under the library for which they're registered. This might include files that your app isn't interested in or can't process. In that case, applying a restrictive filter as soon as the background task is triggered is the best way to make sure there's no wasted background processing.

Finding the most recently modified or added files is as easy as a single query against the indexer. Simply request all the files with a gather time falling in the range in which the app is interested. The same sorting and grouping features available for other queries can be used here, as well, if desired. Be aware that the indexer uses Zulu time internally, so make sure to convert all the time strings to Zulu before using them. Here's how a query can be built:

```
QueryOptions options = new QueryOptions();
DateTimeOffset lastSearchTime = DateTimeOffset.UtcNow.AddHours(-1);
// This is the conversion to Zulu time, which is used by the indexer
string timeFilter = "System.Search.GatherTime:>=" +
  lastSearchTime.ToString("yyyy\\-MM\\-dd\\THH\\:mm\\:ss\\Z")
options.ApplicationSearchFilter += timeFilter;
```

In this case the app is going to get all the results from the past hour. In some apps it makes more sense to save the time of the last query and use it in the query instead, but any DateTimeOffset will work. Once the app has the list of files back it can enumerate them as discussed earlier or use the list to track which files are new to it.

Combining the two methods of change tracking with the gather time allows UWP apps the ability to change track the file system and react to changes on the disk with ease. These may be relatively new APIs in the history of Windows, but they've been used in the Photos, Groove Music, OneDrive, Cortana, and Movies & TVs apps built into Windows 10. You can feel confident including them in your app knowing that they're powering these great experiences.

General Best Practices

There are a few things that any app using the indexer should be aware of to avoid any bugs and make sure the app is as fast as possible. They include avoiding any unnecessarily complex queries in performance-critical parts of the app, using properties enumerations correctly and being aware of indexing delays.

How a query is designed can have a significant impact on its performance. When the indexer is running queries against its backing database, some queries are faster because of how the information is laid out on the disk. Filtering based off file location is always fast as the indexer is able to quickly eliminate massive parts of the index from the query. This saves processor and I/O time because there are fewer strings that need to be retrieved and compared while searching for matches to query terms.

The indexer is powerful enough to handle regular expressions, but some forms of them are notorious for causing slowdowns. The worst thing that can be included in an indexer query is a suffix search. This is a query for all terms ending with a given value. An example would be the query "*tion," which looks for all documents containing words ending with "tion." Because the index is sorted by the first letter in each token, there's no fast way to find terms matching this query. It has to decode every token in the entire index and compare it to the search term, which is extremely slow.

Enumerations can accelerate queries but have unexpected behavior in international builds. Anyone who has built a search system knows how much faster doing comparisons based off an enumeration is than doing string comparisons. And this is true in the indexer, as well. To make it easy for your app, the property system provides a number of enumerations to filter results to fewer items before starting the costly string comparisons. A common example of this is to use the System.Kind filter to restrict the result to just file kinds that an app can handle, such as music or documents.

There's a common error of which anyone using enumerations must be aware. If your user is going to be looking for only music files, then in an en-us version of Windows adding System.Kind:=music to the query is going to work perfectly to restrict the search results and speed up a query. This will also work in some other languages—possibly even enough to pass internationalization testing—but it will fail where the system isn't able to recognize "music" as an English term and instead parses it in the local language.

The correct way to use an enumeration such as System.Kind is to clearly denote that the app is intending to use the value as an enumeration, not as a search term. This is done by using the enumeration#value syntax. For example, the correct way to filter to just music results would be to write System.Kind:=System.Kind#Music. This will work in all the languages that Windows ships and will filter the results to just files the system recognizes as music files.

Figure 4 **Results with and Without Prefetch**

Test Case (2,600 Images on a Desktop Machine)	Average Run Time More Than 10 Samples
Naïve code + indexer	9,318ms
All optimizations + indexer	5,796ms
Optimizations + no indexer	20,248ms (48,420ms cold)

Correctly escaping Advanced Query Syntax (AQS) can help make sure your users don't hit hard to repro query issues. AQS has a number of features that let users include quotes or parentheses to affect how the query is processed. This means that apps have to be careful to escape any query terms that might include these characters. For example, searching for Document(8).docx is going to result in a parsing error and incorrect results being returned. Instead the app should escape the term as Document%288%29.docx. This will return items in the index that match the search term, rather than having the system try to parse the parenthesis as a part of the query.

For the most in-depth coverage of all the different features of AQS and how to make sure your queries are correct, you can check out the documentation at bit.ly/1Fhacfl. It has a lot of great information, including more details about the tips mentioned here.

A note about indexing delays: Indexing isn't instant, which means that items appearing in the index or notifications based on the indexer are going to be delayed slightly from when the file is written. Under normal system load, the delay is going to be on the order of 100ms, which is faster than most apps can query the file system, so it won't be noticeable. There are cases where a user might be migrating thousands of files around their machine and the indexer is falling noticeably behind.

In these cases, there are two things apps are recommended to do: First, they should hold a query open over the file system locations in which the app is most interested. Typically, this is done by creating a StorageFileQueryResult object over the file system locations the app is going to be searching. When the indexer sees that an app has an open query, it's going to prioritize indexing in those scopes over all other scopes. But please make sure not to do this for a larger scope than needed. The indexer is going to stop respecting system backoff and user-active notifications to process the changes as fast as it can, so users might notice an impact on system performance while this is happening.

The other recommendation is to warn the users that the system is catching up with the file operations. Some apps such as Cortana show a message at the top of their UI, whereas others will stop doing complex queries and show a simple version of the experience. It's up to you to determine what's best for your app experience.

Wrapping Up

This has been a quick tour of the features that are available for consumers of the indexer and Windows Storage APIs in Windows 10. For more information about how to use queries to pass context on app activation or code samples for the background trigger, check out the team's blog at bit.ly/1iPUVIo. We're constantly working with developers to make sure that the searching APIs are great to use. We would love to hear your feedback about what's working and what you'd like to see added to the surface. ■

Adam Wilson *is a program manager on the Windows Developer Ecosystem and Platform team. He works on the Windows indexer and push notification reliability. Previously he worked on the storage APIs for Windows Phone 8.1. Reach him at adwilso@microsoft.com.*

Thanks to the following technical expert for reviewing this article:
Sami Khoury

Async from the Start

Mark Sowul

Recent versions of the Microsoft .NET Framework make it easier than ever to write responsive, high-performance applications via the async and await keywords—it's no exaggeration to say that they've changed the way we .NET developers write software. Asynchronous code that used to require an impenetrable web of nested callbacks can now be written (and understood!) almost as easily as sequential, synchronous code.

There's ample material already on creating and consuming async methods, so I'll assume you're familiar with the basics. If you aren't, the Visual Studio Documentation page at `msdn.com/async` can get you up to speed.

Most of the documentation about async warns that you can't just plug an async method into existing code; the caller itself needs to be async. In the words of Lucian Wischik, a developer on the Microsoft language team, "Async is like the zombie virus." So how do you build async into the very fabric of your application, right from the beginning, without resorting to async void? I'm going to show you over the course of several refactorings of the default UI startup code, both for Windows Forms and Windows Presentation Foundation (WPF), transforming the UI boilerplate into an object-oriented design and adding support for async/await. Along the way, I'll also explain when it does and doesn't make sense to use "async void."

In this article, the main walk-through focuses on Windows Forms; WPF requires additional changes, which can get distracting. In each step, I'll first explain the changes with a Windows Forms application, and then discuss any differences needed for the WPF version. I show all the basic code changes in the article, but you can see completed examples (and the intermediate revisions) for both environments in the accompanying online code download.

This article discusses:

- Refactoring the default Windows Forms and Windows Presentation Foundation startup code to be object-oriented
- Decoupling an application's lifetime from the form/window
- Making startup code asynchronous
- Handling exceptions thrown from asynchronous code
- Adding a splash screen without a new thread

Technologies discussed:

Windows Forms, Windows Presentation Foundation, Async and Await Keywords

Code download available at:

msdn.com/magazine/msdnmag1115

First Steps

The Visual Studio templates for Windows Forms and WPF applications don't really lend themselves to using async during startup (or to customizing the startup process in general). Although C# strives to be an object-oriented language—all code has to be in classes—the default startup code nudges developers toward placing logic in static methods alongside Main, or in an overly complicated constructor for the main form. (No, it's not a good idea to access the database inside the MainForm constructor. And, yes, I've seen it done.) This situation has always been problematic, but now with async, it also means there's no clear opportunity to have the application initialize itself asynchronously.

To start, I created a new project with the Windows Forms Application template in Visual Studio. **Figure 1** shows its default startup code in Program.cs.

It's not as easy with WPF. The default WPF startup is quite opaque, and even finding any code to customize is difficult. You can put some initialization code in Application.OnStartup, but how would you delay showing the UI until you've loaded the necessary data? The first thing I need to do with WPF is expose the startup process as code I can edit. I'll get WPF to the same starting point as Windows Forms, and then each step of the article is similar for both.

After creating a new WPF application in Visual Studio, I create a new class, called Program, with the code in **Figure 2**. To replace the default startup sequence, open the project properties and change the startup object from "App" to the newly created "Program."

Figure 1 **The Default Windows Forms Startup Code**

```
static class Program
{
  /// <summary>
  /// The main entry point for the application.
  /// </summary>
  [STAThread]
  static void Main()
  {
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);
    Application.Run(new Form1());
  }
}
```

Figure 2 **The Equivalent Windows Presentation Foundation Startup Code**

```
static class Program
{
  /// <summary>
  /// The main entry point for the application.
  /// </summary>
  [STAThread]
  static void Main()
  {
    App app = new App();
    // This applies the XAML, e.g. StartupUri, Application.Resources
    app.InitializeComponent();
    // Shows the Window specified by StartupUri
    app.Run();
  }
}
```

If you use "Go to Definition" on the call to InitializeComponent in **Figure 2**, you'll see the compiler generates the equivalent Main code as when you use App as the startup object (which is how I can open the "black box" for you here).

Toward an Object-Oriented Startup

First, I'll do a small refactoring of the default startup code to push it in an object-oriented direction: I'll take the logic out of Main and move it to a class. To do that, I'll make Program a non-static class (as I said, the defaults push you in the wrong direction) and give it a constructor. Then I'll move the setup code to the constructor, and add a Start method that will run my form.

I've called the new version Program1, and you can see it in **Figure 3**. This skeleton shows the core of the idea: to run the program, Main now creates an object and calls methods on it, just as with any typical object-oriented scenario.

Decoupling the Application from the Form

Nevertheless, that call to Application.Run that takes a form instance (at the end, in my Start method) poses a few problems. One is a generic architectural concern: I don't like that it ties my application's lifetime to displaying that form. This would be OK for many applications, but there are applications that run in the background that should not display any UI when they start, except maybe an icon in the taskbar or the notification area. I know I've seen some that briefly flash a screen when they launch, before disappearing. My bet is that their startup code follows a similar process, and then they hide themselves as soon as possible when the form is finished loading. Admittedly, that particular problem isn't necessary to solve here, but the separation will be of critical importance for initializing asynchronously.

Instead of Application.Run(m_mainForm), I'll use the overload of Run that doesn't take an argument: It starts the UI infrastructure without tying it to any particular form. This decoupling means I have to show the form myself; it also means that closing the form will no longer quit the app, so I need to wire that up explicitly, too, as shown in **Figure 4**. I'll also use this opportunity to add my first hook for initialization. "Initialize" is a method I'm creating on my form class to hold any logic I need for initializing it, such as retrieving data from a database or a Web site.

In the WPF version, the app's StartupUri determines what window to show when Run is called; you'll see it defined in the App.xaml markup file. Unsurprisingly, the Application default ShutdownMode setting of OnLastWindowClose shuts down the application when all the WPF windows have closed, so that's how the lifetimes get tied together. (Note that this differs from Windows Forms. In Windows Forms, if your main window opens a child window and you close just the first window, the application will exit. In WPF, it won't exit unless you close both windows.)

To accomplish the same separation in WPF, I first remove the StartupUri from App.xaml. Instead, I create the window myself, initialize it and show it before the call to App.Run:

```
public void Start()
{
  MainWindow mainForm = new MainWindow();
  mainForm.Initialize();
  mainForm.Show();
  m_app.Run();
}
```

When I create the application, I set app.ShutdownMode to ShutdownMode.OnExplicitShutdown, which decouples the application lifetime from that of the windows:

```
m_app = new App();
m_app.ShutdownMode = ShutdownMode.OnExplicitShutdown;
m_app.InitializeComponent();
```

To accomplish that explicit shutdown, I'll attach an event handler for MainWindow.Closed.

Figure 3 **Program1, the Beginning of an Object-Oriented Startup**

```
[STAThread]
static void Main()
{
  Program1 p = new Program1();
  p.Start();
}

private readonly Form1 m_mainForm;
private Program1()
{
  Application.EnableVisualStyles();
  Application.SetCompatibleTextRenderingDefault(false);

  m_mainForm = new Form1();
}

public void Start()
{
  Application.Run(m_mainForm);
}
```

Figure 4 **Program2, the Message Loop Is Now Separate from the Main Form**

```
private Program2()
{
  Application.EnableVisualStyles();
  Application.SetCompatibleTextRenderingDefault(false);

  m_mainForm = new Form1();
  m_mainForm.FormClosed += m_mainForm_FormClosed;
}

void m_mainForm_FormClosed(object sender, FormClosedEventArgs e)
{
  Application.ExitThread();
}

public void Start()
{
  m_mainForm.Initialize();
  m_mainForm.Show();
  Application.Run();
}
```

Of course, WPF does a better job of separating concerns, so it makes more sense to initialize a view model rather than the window itself: I'll create a MainViewModel class and create my Initialize method there. Similarly, the request to close the app should also go through the view model, so I'll add a "CloseRequested" event and a corresponding "RequestClose" method to the view model. The resulting WPF version of Program2 is listed in **Figure 5** (Main is unchanged, so I won't show it here).

Pulling out the Hosting Environment

Now that I've separated Application.Run from my form, I want to handle another architectural consideration. Right now, Application is deeply embedded in the Program class. I want to "abstract out" this hosting environment, so to speak. I'm going to remove all the various Windows Forms methods on Application from my Program class, leaving only the functionality related to the program itself, as shown with Program3 in **Figure 6**. One last piece is to add an event on the program class so the link between closing the form and shutting down the application is less direct. Notice how Program3 as a class has no interaction with Application!

Separating the hosting environment has a few benefits. For one, it makes testing easier (you can now test Program3, to a limited extent). It also makes it easier to reuse the code elsewhere, perhaps embedded into a larger application or a "launcher" screen.

The decoupled Main is shown in **Figure 7**—I've moved the Application logic back to it. This design makes it easier to integrate WPF and Windows Forms, or perhaps to gradually replace Windows Forms with WPF. That's outside the scope of this article, but you can find an example of a mixed application in the accompanying online code. As with the prior refactoring, these are nice things but not necessarily crucial: The relevance to the "Task at hand," so to speak, is that it's going to make the asynchronous version flow more naturally, as you'll soon see.

Figure 5 **The Program2 Class, Windows Presentation Foundation Version**

```
private readonly App m_app;
private Program2()
{
  m_app = new App();
  m_app.ShutdownMode = ShutdownMode.OnExplicitShutdown;
  m_app.InitializeComponent();
}

public void Start()
{
  MainViewModel viewModel = new MainViewModel();
  viewModel.CloseRequested += viewModel_CloseRequested;
  viewModel.Initialize();

  MainWindow mainForm = new MainWindow();
  mainForm.Closed += (sender, e) =>
  {
    viewModel.RequestClose();
  };

  mainForm.DataContext = viewModel;
  mainForm.Show();
  m_app.Run();
}

void viewModel_CloseRequested(object sender, EventArgs e)
{
  m_app.Shutdown();
}
```

Long-Awaited Asynchrony

Now, finally, the payoff. I can make the Start method asynchronous, which lets me use await and make my initialization logic asynchronous. As per convention, I've renamed Start to StartAsync, and Initialize to InitializeAsync. I've also changed their return type to async Task:

```
public async Task StartAsync()
{
  await m_mainForm.InitializeAsync();
  m_mainForm.Show();
}
```

To use it, Main changes like so:

```
static void Main()
{
  ...
  p.ExitRequested += p_ExitRequested;
  Task programStart = p.StartAsync();

  Application.Run();
}
```

In order to explain how this works—and solve a subtle but important problem—I need to explore in detail what's going on with async/await.

The true meaning of await: Consider the StartAsync method I presented. It's important to realize that (typically), when an async method reaches the await keyword, it returns. The executing thread continues, just as it would when any method returns. In this case, the StartAsync method reaches "await m_mainForm.InitializeAsync" and returns to Main, which continues, calling Application.Run. This leads to the somewhat counterintuitive result that Application.Run is likely to execute before m_mainForm.Show, even though sequentially it occurs after m_mainForm.Show. Async and await do make asynchronous programming easier, but it's still by no means easy.

That's why async methods return Tasks; it's that Task completing that represents the async method "returning" in the intuitive sense, namely when all of its code has run. In the case of StartAsync, it means that it has completed both InitializeAsync and m_mainForm.Show. And this is the first problem with using async void: Without a task object, there's no way for the caller of an async void method to know when it has finished executing.

How and when does the rest of the code run, if the thread has moved on and StartAsync has already returned to its caller? This is where Application.Run comes in. Application.Run is an infinite

Figure 6 **Program3, Now Easy to Plug in Elsewhere**

```
private readonly Form1 m_mainForm;
private Program3()
{
  m_mainForm = new Form1();
  m_mainForm.FormClosed += m_mainForm_FormClosed;
}

public void Start()
{
  m_mainForm.Initialize();
  m_mainForm.Show();
}

public event EventHandler<EventArgs> ExitRequested;
void m_mainForm_FormClosed(object sender, FormClosedEventArgs e)
{
  OnExitRequested(EventArgs.Empty);
}

protected virtual void OnExitRequested(EventArgs e)
{
  if (ExitRequested != null)
    ExitRequested(this, e);
}
```

loop, waiting for work to do—mainly processing UI events. For example, when you move your mouse over the window, or click a button, the Application.Run message loop will dequeue the event and dispatch the appropriate code in response, and then wait for the next event to come in. It's not strictly limited to the UI, though: Consider Control.Invoke, which runs a function on the UI thread. Application.Run is processing these requests, too.

In this case, once InitializeAsync completes, the remainder of the StartAsync method will be posted to that message loop. When you use await, Application.Run will execute the rest of the method on the UI thread, just as if you'd written a callback using Control.Invoke. (Whether the continuation should occur on the UI thread is controlled by ConfigureAwait. You can read more about that in Stephen Cleary's March 2013 article on best practices in asynchronous programming at msdn.com/magazine/jj991977).

This is why it was so important to separate Application.Run from m_mainForm. Application.Run is running the show: it needs to be running in order to process the code after the "await," even before you're actually showing any UI. For example, if you try moving Application.Run out of Main and back into StartAsync, the program will just quit immediately: Once execution hits "await InitializeAsync," control returns to Main, and then there's no more code to run, so that's the end of Main.

This also explains why the use of async has to start from the bottom up. A common but short-lived antipattern is to call Task.Wait instead of await, because the caller isn't an async method, but most likely it will deadlock immediately. The problem is that the UI thread will be blocked by that call to Wait and unable to process the continuation. Without the continuation, the task won't complete, so the call to Wait will never return—deadlock!

Await and Application.Run, a chicken and an egg problem: I mentioned earlier there was a subtle problem. I described that when you call await, the default behavior is to continue execution on the UI thread, which is what I need here. However, the infrastructure for that isn't set up when I first call await, because the appropriate code hasn't run yet!

SynchronizationContext.Current is the key to this behavior: When calling await, the infrastructure captures the value of SynchronizationContext.Current, and uses that to post the continuation; that's how it continues on the UI thread. The synchronization context is set up by Windows Forms or WPF when it starts running the message loop. Inside of StartAsync, that hasn't happened yet: If you examine SynchronizationContext.Current in the beginning of StartAsync, you'll see it's null. If there's no synchronization context, await will post the continuation to the thread pool instead, and because that's not going to be the UI thread, it's not going to work.

The WPF version will hang outright, but, as it turns out, the Windows Forms version will "accidentally" work. By default, Windows Forms sets up the synchronization context when the first control is created—in this case, when I construct m_mainForm (this behavior is controlled by WindowsFormsSynchronizationContext.AutoInstall). Because "await InitializeAsync" occurs after I create the form, I'm OK. Were I to put an await call *before* creating m_mainForm, however, I'd have the same problem. The solution is to set up the synchronization context myself in the beginning, as follows:

```
[STAThread]
static void Main()
{
  Application.EnableVisualStyles();
  Application.SetCompatibleTextRenderingDefault(false);
  SynchronizationContext.SetSynchronizationContext(
    new WindowsFormsSynchronizationContext());

  Program4 p = new Program4();
  ... as before
}
```

For WPF, the equivalent call is:

```
SynchronizationContext.SetSynchronizationContext(
  new DispatcherSynchronizationContext());
```

Exception Handling

Almost there! But I still have another lingering problem at the root of the application: If InitializeAsync raises an exception, the program doesn't handle it. The programStart task object will contain the exception information, but nothing is being done with it and so my application will be hung in a sort of purgatory. If I could "await StartAsync," I could catch the exception in Main, but I can't use await, because Main isn't async.

This illustrates the second problem with async void: There's no way to properly catch exceptions thrown by an async void method because the caller has no access to the task object. (So when *should* you use async void? Typical guidance says async void should be limited mostly to event handlers. The March 2013 article I mentioned before discusses this, too; I recommend reading it to get the most out of async/await.)

Under normal circumstances, TaskScheduler.UnobservedException deals with tasks that raise exceptions that aren't subsequently handled. The problem is, it's not guaranteed to run. In this situation, it almost certainly won't: the task scheduler detects unobserved exceptions when such a task is finalized. Finalization happens only when the garbage collector runs. The garbage collector runs only when it needs to satisfy a request for more memory.

You may see where this is going: In this case, an exception will result in the application sitting around doing nothing, so it won't request more memory, so the garbage collector won't run. The effect is that the app will hang. In fact, that's why the WPF version hangs if you don't specify the synchronization context: the WPF window constructor throws an exception because a window is being created on a non-UI thread, and then that exception goes unhandled. A final piece, then, is to deal with the programStart task, and add a continuation that will run in case of error. In this case, it makes sense to quit if the application can't initialize itself.

Figure 7 **Main, Now Able to Host an Arbitrary Program**

```
[STAThread]
static void Main()
{
  Application.EnableVisualStyles();
  Application.SetCompatibleTextRenderingDefault(false);

  Program3 p = new Program3();
  p.ExitRequested += p_ExitRequested;
  p.Start();

  Application.Run();
}

static void p_ExitRequested(object sender, EventArgs e)
{
  Application.ExitThread();
}
```

I can't use await in Main because it isn't async, but I can create a new async method solely for the purpose of exposing (and handling) any exceptions thrown during the asynchronous startup: It will consist only of a try/catch around an await. Because this method will be handling all exceptions and not throwing any new ones, it's another of the limited cases where async void makes sense:

```
private static async void HandleExceptions(Task task)
{
  try
  {
    await task;
  }
  catch (Exception ex)
  {
    ...log the exception, show an error to the user, etc.
    Application.Exit();
  }
}
```

Main uses it as follows:

```
Task programStart = p.StartAsync();
HandleExceptions(programStart);
Application.Run();
```

Of course, as usual, there's a subtle issue (if async/await makes things easier, you can imagine how hard it used to be). I said earlier that *typically*, when an async method reaches a call to await, it returns, and the remainder of that method runs as a continuation. In some cases, though, the task can complete synchronously; if that's the case, the execution of the code doesn't get broken up, which is a performance benefit. If that happens here, though, it means that the HandleExceptions method will run in its entirety and then return, and Application.Run will follow it: In that case, if there is an exception, now the call to Application.Exit will occur *before* the call to Application.Run, and it won't have any effect.

What I want to do is force HandleExceptions to run as a continuation: I need to make sure that I "fall through" to Application.Run before doing anything else. This way, if there's an exception, I know that Application.Run is already executing, and Application.Exit will properly interrupt it. Task.Yield does just that: It forces the current async code path to yield to its caller, and then resume as a continuation.

Here is the correction to HandleExceptions:

```
private static async void HandleExceptions(Task task)
{
  try
  {
    // Force this to yield to the caller, so Application.Run will be executing
    await Task.Yield();
    await task;
  }
  ...as before
```

In this case, when I call "await Task.Yield", HandleExceptions will return and Application.Run will execute. The remainder of HandleExceptions will then be posted as a continuation to the current SynchronizationContext, which means it will be picked up by Application.Run.

Incidentally, I think Task.Yield is a good litmus test for understanding async/await: If you understand the use of Task.Yield, then you probably have a solid understanding of how async/await works.

The Payoff

Now that everything is working, it's time to have some fun: I'm going to show how easy it is to add a responsive splash screen without running it on a separate thread. Fun or not, having a splash screen is quite important if your application doesn't "start" right away: If the user launches your application and doesn't see anything happen for several seconds, that's a bad user experience.

Starting a separate thread for a splash screen is inefficient, and it's also clunky—you have to marshal all the calls properly between threads. Providing progress information on the splash screen is therefore difficult, and even closing it requires a call to Invoke or the equivalent. Moreover, when the splash screen finally does close, usually it doesn't properly give focus to the main form, because it's impossible to set the ownership between the splash screen and the main form if they're on different threads. Compare that to the simplicity of the asynchronous version, shown in **Figure 8**.

Figure 8 **Adding a Splash Screen to StartAsync**

```
public async Task StartAsync()
{
  using (SplashScreen splashScreen = new SplashScreen())
  {
    // If user closes splash screen, quit; that would also
    // be a good opportunity to set a cancellation token
    splashScreen.FormClosed += m_mainForm_FormClosed;
    splashScreen.Show();

    m_mainForm = new Form1();
    m_mainForm.FormClosed += m_mainForm_FormClosed;
    await m_mainForm.InitializeAsync();

    // This ensures the activation works so when the
    // splash screen goes away, the main form is activated
    splashScreen.Owner = m_mainForm;
    m_mainForm.Show();

    splashScreen.FormClosed -= m_mainForm_FormClosed;
    splashScreen.Close();
  }
}
```

Wrapping Up

I have shown how to apply an object-oriented design to your application's startup code—whether Windows Forms or WPF—so it can easily support asynchronous initialization. I've also shown how to overcome some subtle problems that can come from an asynchronous startup process. As for actually making your initialization asynchronous, I'm afraid you're on your own for that, but you'll find some guidance at msdn.com/async.

Enabling the use of async and await is just the start. Now that Program is more object-oriented, other features become more straightforward to implement. I can process command-line arguments by calling an appropriate method on the Program class. I can have the user log in before showing the main window. I can start the app in the notification area without showing any window at startup. As usual, an object-oriented design provides the opportunity to extend and reuse functionality in your code. ■

Mark Sowul *may in fact be a software simulation written in C# (so folks speculate). A devoted .NET developer since the beginning, Sowul shares his wealth of architecture and performance expertise in .NET and Microsoft SQL Server via his New York consulting business, SolSoft Solutions. Reach him at mark@solsoftsolutions.com, and sign up for his occasional e-mails on software insights at eepurl.com/_K7YD.*

Thanks to the following Microsoft technical experts for reviewing this article: *Stephen Cleary and James McCaffrey*

Use ASP.NET as a High-Performance File Downloader

Doug Duerner and Yeon-Chang Wang

Slow and faulty connections have long been the bane of large file downloads. You can be in an airport concourse gathering media over a sketchy WiFi connection to work on a presentation during a long flight, or on the African savannah trying to download a large installation file via satellite link for a solar-powered water pump. In either instance, the cost of having a large file download crash is the same: time lost, productivity drained and the success of the assignment imperiled.

It doesn't have to be that way. In this article we show how to create a utility to address the problem of resuming and continuing failed downloads that can be caused by poor connections that are prone to dropping offline during large file transfers.

This article discusses:

- Byte ranges and partial content in the HTTP protocol
- Dividing a file into pieces and downloading the pieces on separate threads
- Retrying only pieces that failed to download instead of starting over
- Writing file pieces directly to response stream to keep server memory usage low
- Possible future enhancements for extremely rudimentary mirror server infrastructure

Technologies discussed:

ASP.NET, IIS Web Server, HTTP Protocol

Code download available at:

msdn.com/magazine/msdnmag1115

Background

We wanted to create a simple file downloader utility that could easily be added to your existing IIS Web Server, with an extremely simple and easy-to-use client program (or the option of merely using the Web browser as a client).

The IIS Web Server has already proven to be a highly scalable, enterprise-grade Web server, serving up files to browsers for years. We basically wanted to take advantage of the IIS Web Server's ability to handle many HTTP Web requests at the same time, in parallel, and apply that to file downloading (copying).

Essentially, we needed a file downloader utility that could download huge files to users around the world who were sometimes located in remote regions with slow and often faulty network links. With the possibility of some remote users around the world still using modem links or faulty satellite links that might be going offline at random times or intermittently toggling between online and offline, the utility would need to be extremely resilient with the ability to retry *only* the portions of the file that failed to download. We didn't want a user to spend all night downloading a huge file over a slow link, and if there was one small hiccup in the network link, need to start the entire download process over again. We also needed to ensure these huge files being downloaded weren't buffered in server memory and that the server memory usage was minimal, so memory usage wouldn't keep rising until server failure when many users were downloading files at the same time.

Conversely, if the user was lucky enough to have a reliable high-speed network link—with both the client and server machines being high-end computers equipped with multiple CPUs and network cards—we wanted the user to be able to download a file using multiple threads and multiple connections, allowing the download of multiple chunks of the file at the same time in parallel using all hardware resources, while at the same time using minimal server memory.

In a nutshell, we created a simple, multithreaded, parallel, low-memory-usage file download utility that can divide the file into chunks, download the individual chunks on separate threads and allow a user to retry only the chunks that failed to download.

The sample project that accompanies this article contains the code for the file download utility and provides a rudimentary base infrastructure that can be expanded going forward, allowing you to get more sophisticated as need arises.

> You can virtually eliminate the negative impact of a faulty connection that's continually going offline.

Sample Project Overview

In essence, DownloadHandler.dll transforms an existing IIS Web Server into a multithreaded file downloader that lets you download a file in chunks, in parallel, using a simple URL from the standalone executable client (FileDownloader.exe), as shown in **Figure 1**. Note that the parameter (chunksize=5242880) is optional, and if not included, will default to download the entire file in one chunk. **Figure 2** and **Figure 3** demonstrate how it allows you to repeatedly retry only the failed portions of the file until they succeed, without

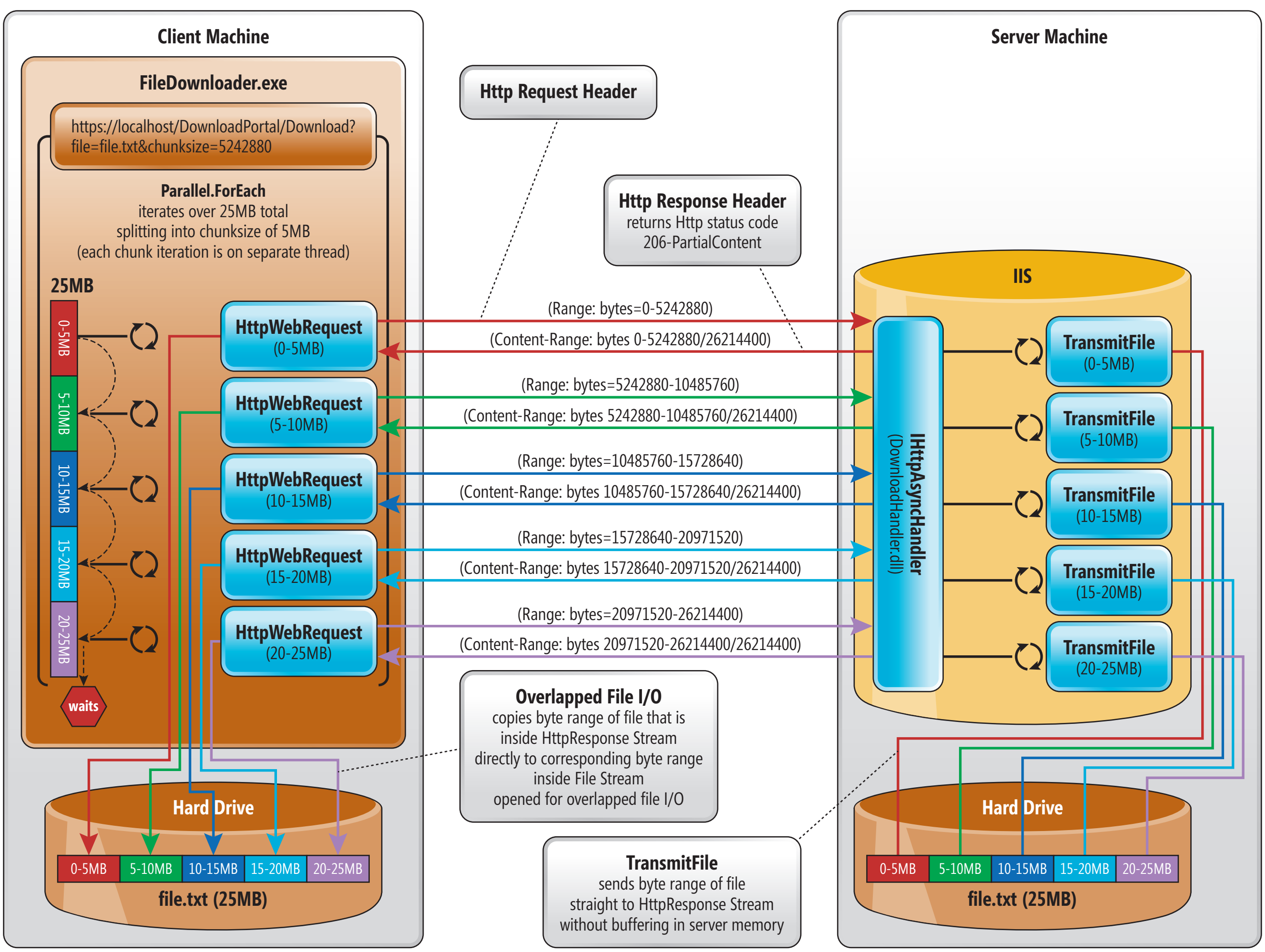


Figure 1 **High-Level Design Overview of Processing Flow for DownloadHandler.dll (Using FileDownloader.exe as Client)**

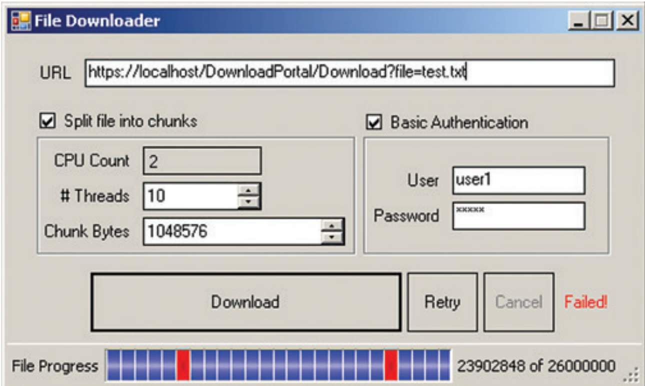


Figure 2 **Standalone Executable as Download Client (with Failed Chunks)**

having to completely restart the entire download from the beginning, like most other file-downloading software.

Figure 1 is a high-level overview of the design of DownloadHandler.dll and FileDownloader.exe, showing the processing flow as the chunks of the file on the server machine's hard drive pass through DownloadHandler.dll and FileDownloader.exe into the file on the client machine's hard drive, illustrating the HTTP protocol headers involved in that process.

In **Figure 1**, FileDownloader.exe initiates a file download by calling the server using a simple URL, which contains the name of the file you want to download as a URL query string parameter (file=file.txt), and internally uses the HTTP method (HEAD), so that initially the server will send back only its response headers, one of which contains the total file size. The client then uses a Parallel.ForEach construct to iterate, splitting the total file size into chunks (byte ranges) based on the chunk size in the parameter (chunksize=5242880). For each individual iteration, the Parallel.ForEach construct executes a processing method on a separate thread, passing in the associated byte range. Inside the processing method, the client issues an HttpWebRequest call to the server using the same URL and internally appends an HTTP request header containing the byte range supplied to that processing method (that is, Range: bytes=0-5242880, Range: bytes=5242880-10485760 and so on).

On the server machine, our implementation of the IHttpAsyncHandler interface (System.Web.IHttpAsyncHandler) handles each request on a separate thread, executing the HttpResponse.TransmitFile method in order to write the byte range requested from the server machine's file directly to the network stream—with no explicit buffering—so the memory impact on the server is almost nonexistent. The server sends back its response with an HTTP Status Code 206 (PartialContent) and internally appends the HTTP response header identifying the byte range being returned (that is, Content-Range: bytes 0-5242880/26214400, Content-Range: bytes 5242880-10485760/26214400 and so on). As each thread receives the HTTP response on the client machine, it writes the bytes returned in the response to the corresponding portion of the file on the client machine's hard drive that was identified in the HTTP response header (Content-Range). It uses asynchronous overlapped file I/O (to ensure the Windows I/O Manager doesn't serialize the I/O requests before dispatching the I/O Request Packets to the kernel-mode driver to complete the file write operation). If multiple user-mode threads all do a file write and you don't have the file opened for asynchronous overlapped I/O, the requests will be serialized and the kernel-mode driver will only receive one request at a time. For more information on asynchronous overlapped I/O, see "Getting Your Driver to Handle More Than One I/O Request at a Time" (`bit.ly/1NIaqxP`) and "Supporting Asynchronous I/O" (`bit.ly/1NIaKMW`) on the Hardware Dev Center site.

To implement the asynchronicity in our IHttpAsyncHandler, we manually post an overlapped I/O structure to the I/O completion port, and the CLR ThreadPool runs the completion delegate supplied in the overlapped structure on a completion port thread. These are the same completion port threads used by most of the built-in async methods. Generally, it's best to use the new built-in async methods for most I/O-bound work, but in this case we wanted to use the HttpResponse.TransmitFile function due to its outstanding ability to transfer huge files without explicitly buffering them in server memory. It's amazing!

Parallel.ForEach is primarily for CPU-bound work and should never really be used in a server implementation due to its blocking nature. We offload the work to a completion port thread from the CLR ThreadPool, instead of a regular worker thread from the CLR ThreadPool, in order to keep from depleting the same threads used by IIS to service incoming requests. Also, the more efficient manner in which the completion port processes work somewhat limits the thread consumption on the server. There's a diagram with a more detailed explanation listed in the sample project code in the comment section at the top of the IOThread class that highlights the differences between the completion port threads and worker threads in the CLR ThreadPool. Because scaling to millions of users isn't the primary goal of this utility, we can afford to expend the additional server threads required to run the HttpResponse.TransmitFile function in order to achieve the associated memory savings on the server when transferring massive files. Essentially, we're trading the loss of scalability caused by using additional threads on the server (instead of the built-in async methods with no threads), in order to use the HttpResponse.TransmitFile function, which consumes extraordinarily minimal server memory. Although it's outside the scope of this article, you could optionally use the built-in async methods in combination with unbuffered file I/O to achieve a similar memory savings with no additional

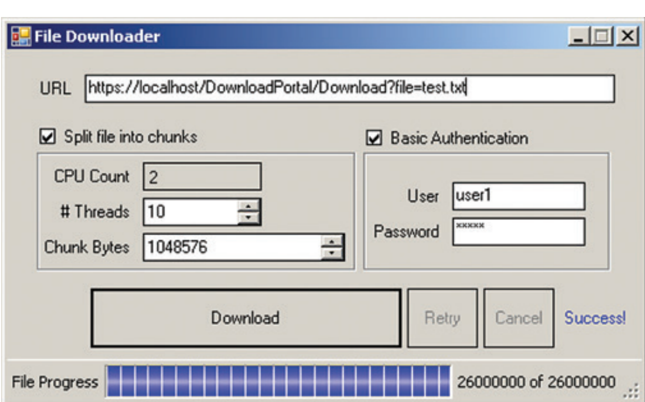


Figure 3 **Standalone Executable as Download Client (After Retry)**

Grid Controls

Blazing fast, feature-complete and just plain beautiful.

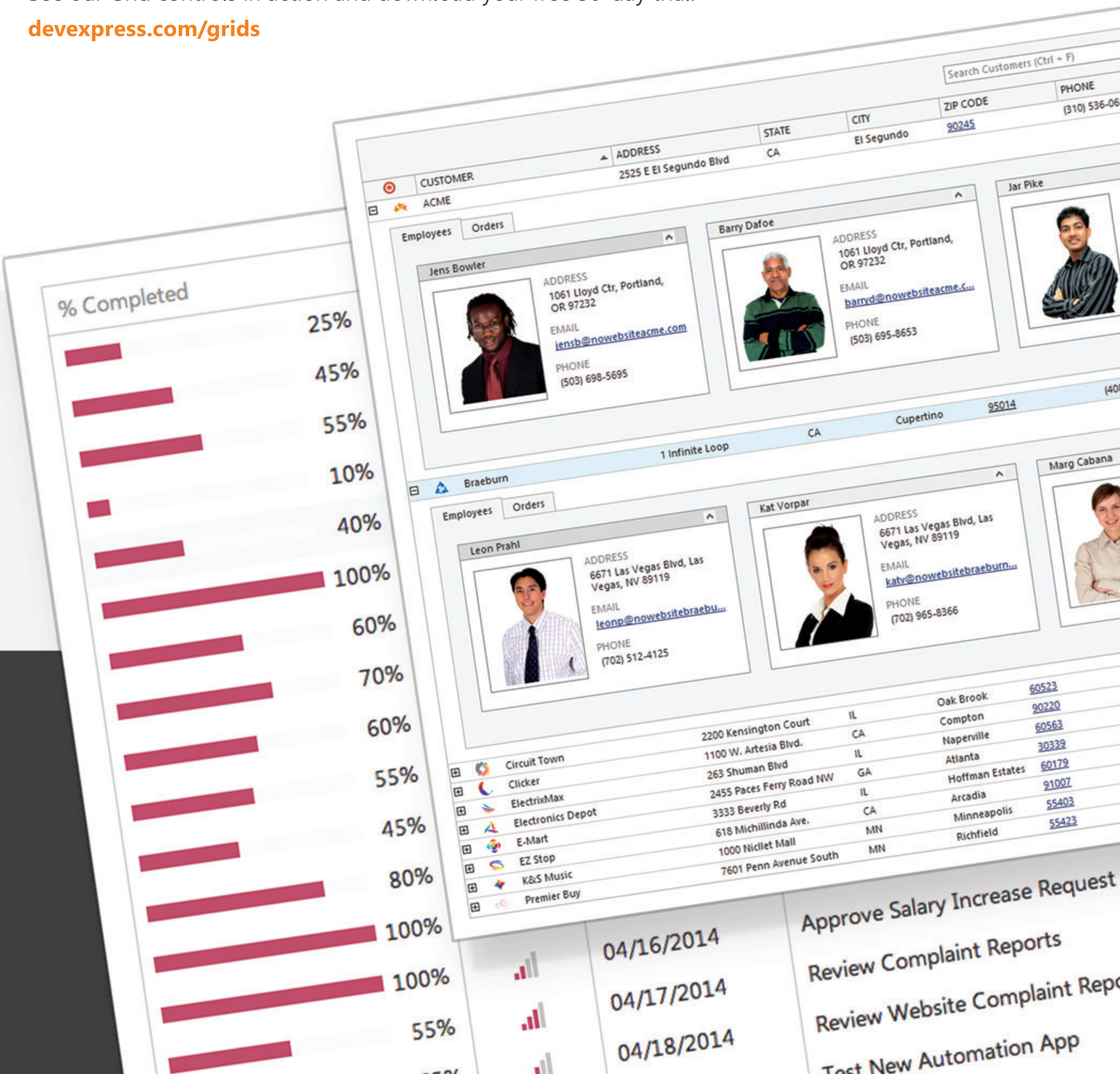
DevExpress Grid controls are Outlook® inspired record editing and data shaping components, allowing your end-users to manage and arrange information on-screen as business requirements dictate. Our data grids ship with dozens of industry leading features including integrated master-detail support and multiple data layout options such as Card and Windows Explorer views. And when you need to target touch enabled devices, all DevExpress Grid controls include a broad range of touch friendly options.

WIN WPF ASP MVC

Your Next Great App Starts @DevExpress

See our Grid controls in action and download your free 30-day trial.

devexpress.com/grids



Reporting

Inform and analyze with absolute ease.

The ever-changing needs of the enterprise requires that reporting platforms offer simple and straightforward end-user customization options so that consumers can freely manipulate report output for maximum clarity. Flexibility is key and ease-of-use paramount.

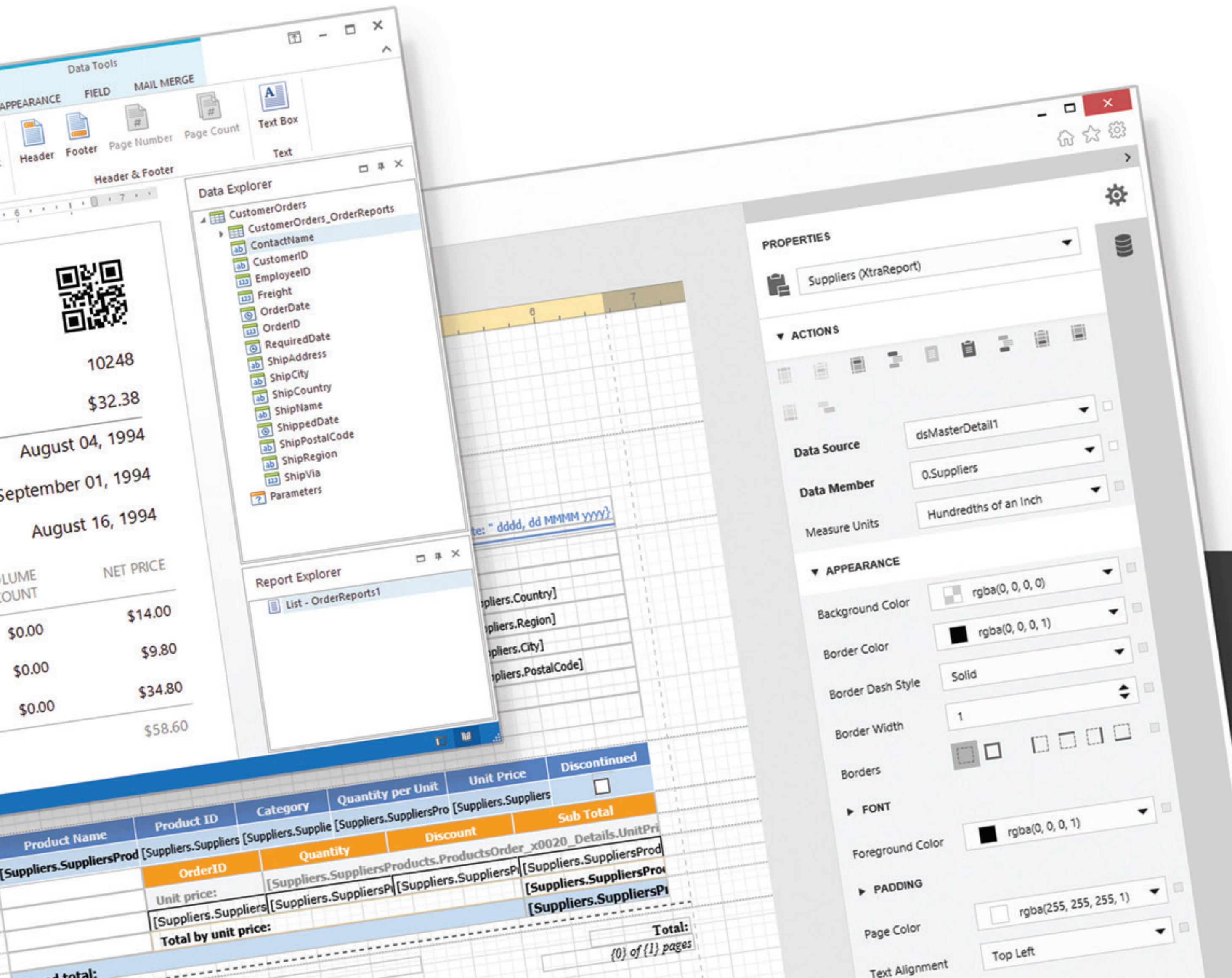
The DevExpress Reporting Suite for .NET was built with your end-users in mind and engineered so you can get the most out of your reporting and data analytics investments. Our Reporting Suite helps you overcome the limitations associated with traditional report tools by providing a fully integrated set of productivity tools, report wizards, pre-built report templates and end-user report designers for both Windows and the web.

WIN WPF ASP MVC

Your Next Great Report Starts @DevExpress

See how to solve your Windows® and web reporting requirements forever.

devexpress.com/reports



Spreadsheets (XLS, XLSx, CSV)

The power of Excel® at your fingertips.

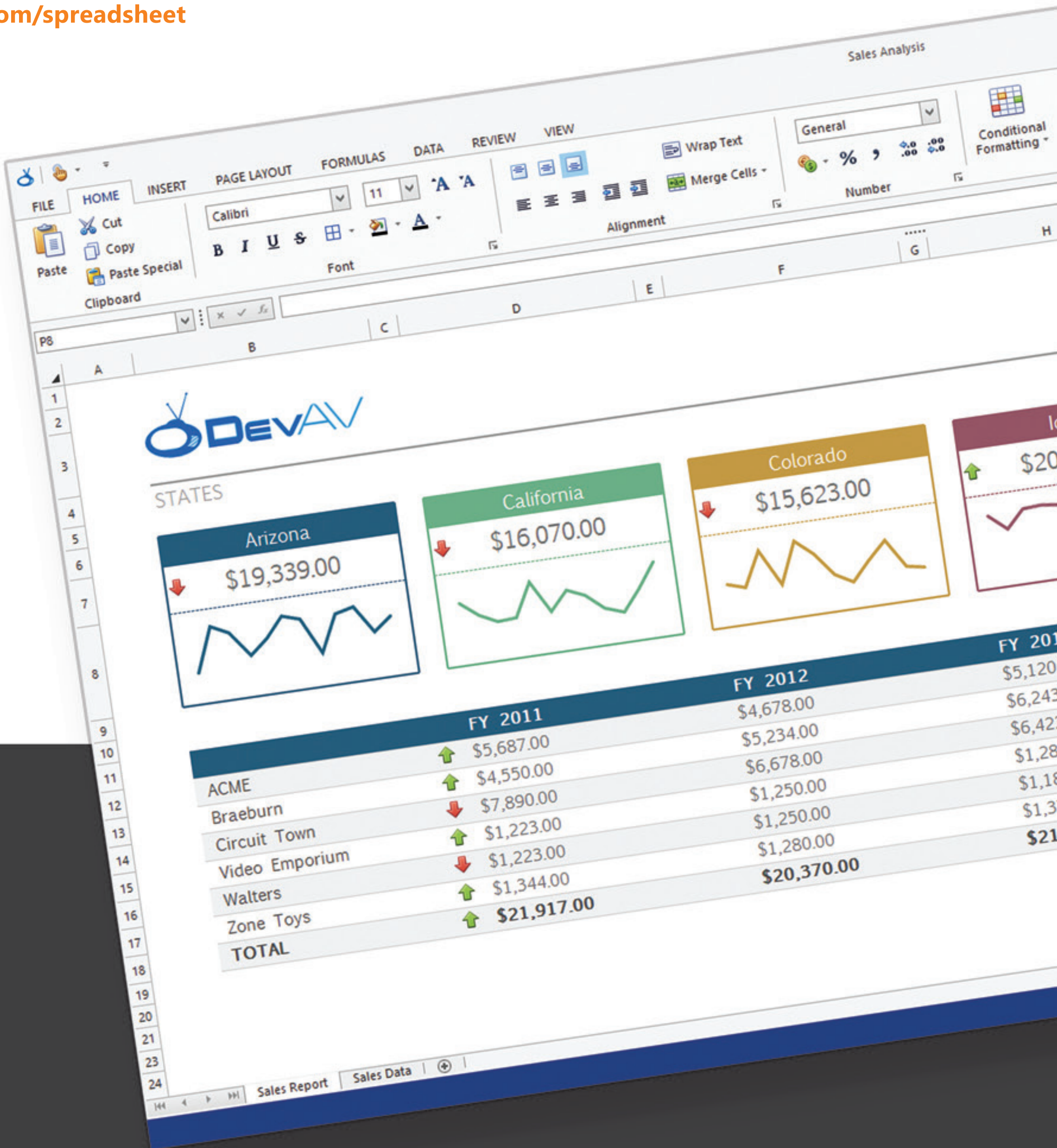
It's no secret...spreadsheets are an important part of business and if you need to incorporate the UX and functionality end-users have come to expect from Microsoft Excel®, DevExpress Spreadsheet is the tool for you. The component library ships with dozens of easy-to-implement features including chart support and fully integrates with other DevExpress components like our Office® inspired Ribbon.

WIN WPF ASP MVC

Your Next Great Spreadsheet Starts @DevExpress

See how you can harness the power of spreadsheets in your next .NET app.

devexpress.com/spreadsheet



Word Processing (DOC, DOCx, RTF)

Create the perfect WYSIWYG document or report.

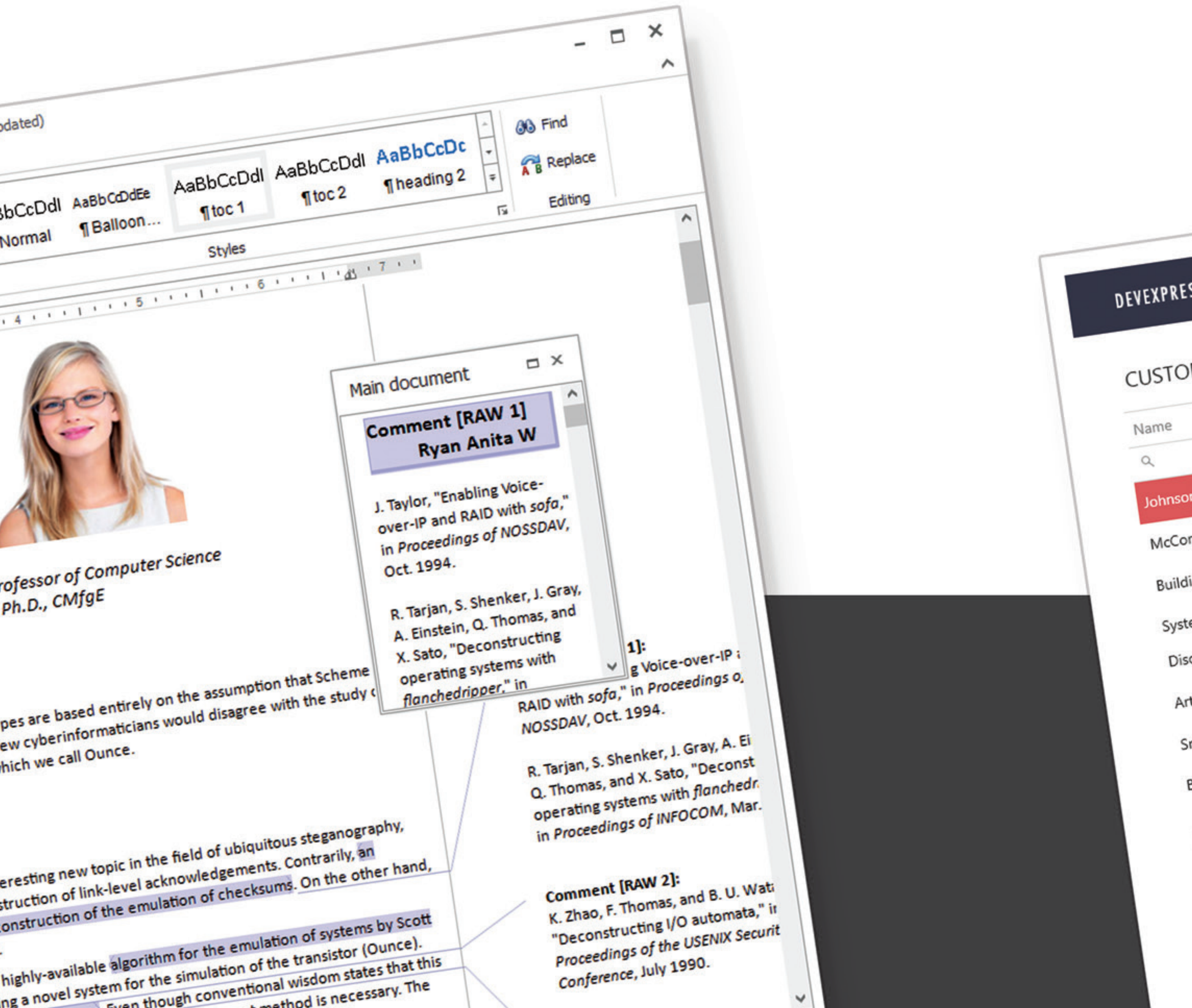
When rich text editing is a must and you need to replicate key features found inside Microsoft Word® (from mail merge and tables, to floating objects and document comments), look no further than the DevExpress Rich Text Editor Control. The library's comprehensive feature set includes printing and PDF export support and the availability of DevExpress Visual Studio project templates means you can incorporate our rich text editors in your next .NET app within minutes of install.

WIN WPF ASP MVC

Your Next Great Text Editor Starts @DevExpress

See how you can introduce Word® compatible rich text editing in your app.

devexpress.com/word



Charting & Analytics

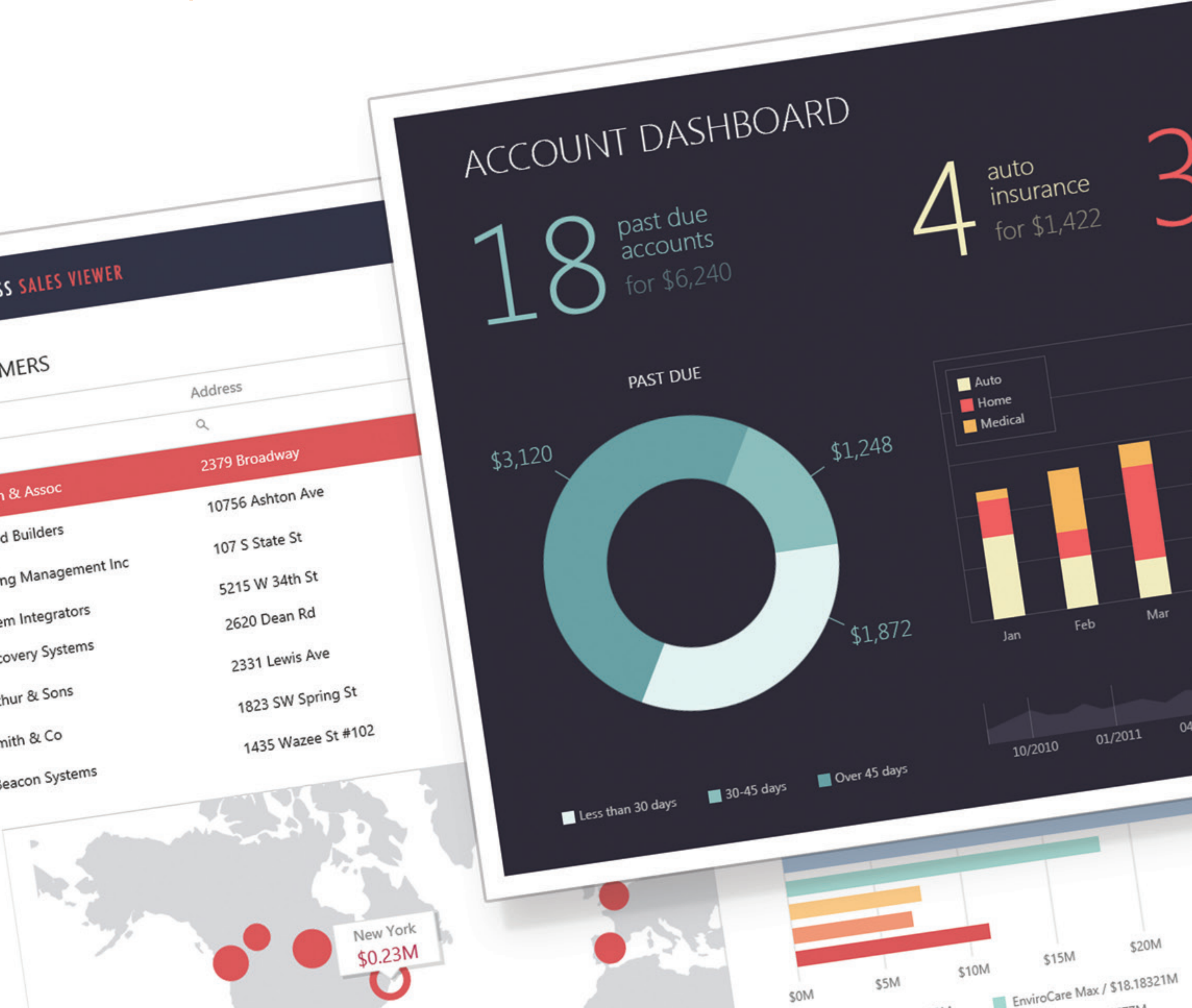
From zero to dashboard in record time.

With our integrated suite of charts, pivot grids, and multi-purpose dashboard widgets, you'll create information-rich decision support systems that are optimized for Windows, the web and your favorite mobile device. The UI components inside our subscriptions help you deliver adaptable, interactive and touch-enabled user experiences that can address a broad range of use-case scenarios. And perhaps best of all, the dashboards you build with DevExpress Universal can be distributed royalty-free.

WIN WPF ASP MVC

Your Next Great Dashboard Starts @DevExpress

See how you can build future proof, enterprise-grade decision support systems.
devexpress.com/dashboard



threads, but from what we understand, everything must be sector aligned and it's somewhat difficult to properly implement. On top of that, it appears Microsoft has purposely removed the NoBuffering item from the FileOptions enum in order to actually prevent unbuffered file I/O, requiring a manual hack to even make it possible. We were quite nervous of the risks associated with not properly implementing it and decided to go with the less risky option of HttpResponse.TransmitFile, which has been fully tested.

FileDownloader.exe can launch multiple threads, each issuing a separate HttpWebRequest call corresponding to a separate portion (byte range) of the file being downloaded based on the total size of the file divided into the "Chunk Bytes" specified, as shown in **Figure 2**.

Any thread that fails to download the portion of the file (byte range) specified in its HttpWebRequest call can be retried by merely making the same HttpWebRequest call (for only that failed byte range) repeatedly until it eventually succeeds, as shown in **Figure 3**. You won't lose the portions of the file already downloaded, which in the case of a slow connection can mean many hours of downloading time saved. You can virtually eliminate the negative impact of a faulty connection that's continually going offline. And with the design's multiple threads downloading different portions of the file at the same time in parallel—directly to the network stream with no explicit buffering—and onto the hard drive with asynchronous overlapped file I/O, you can maximize the amount of downloading accomplished during the window of time when a flaky connection is actually online. The tool will continue to finish the remaining portions each time the network link comes back online, without losing any work. We like to think of it as more of a "retryable" file downloader, not a "resumable" file downloader.

The difference can be illustrated in a hypothetical example. You're going to download a large file that will take all night. You start a resumable file downloader when you leave work and let

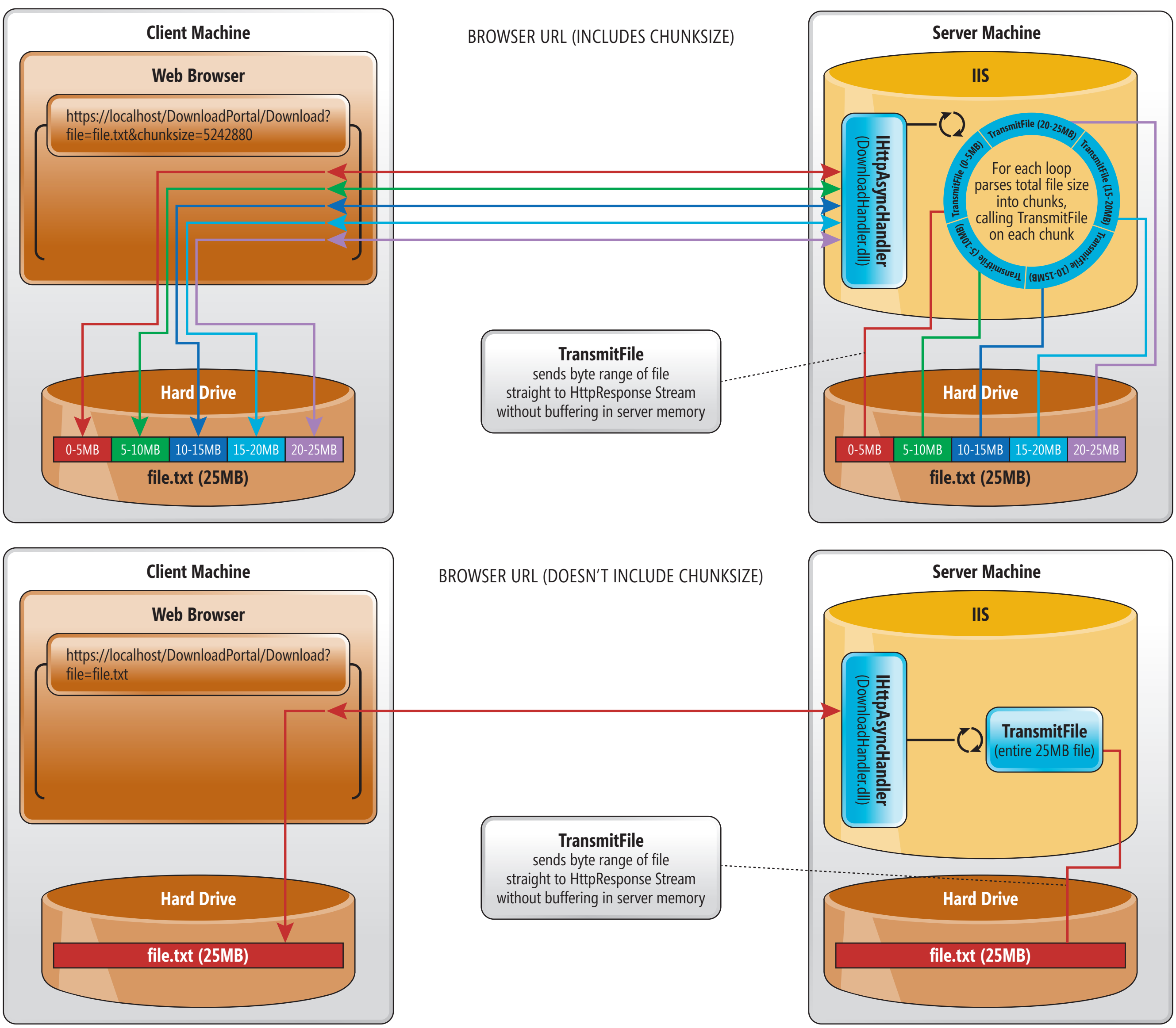


Figure 4 **High-Level Design Overview of Processing Flow for DownloadHandler.dll (Using Web Browser That Doesn't Support Partial Content as Client)**

it run. When you arrive at work in the morning, you see the file download failed at 10 percent and is ready to be resumed. But when it resumes, it will still need to run overnight again in order to finish the remaining 90 percent.

In contrast, you start our retryable file downloader when you leave work and let it run all night. When you arrive at work in the morning, you see the file download failed at *one* chunk at 10 percent, but continued to download the rest of the chunks of the file. Now, you only have to retry just that one chunk and you're done. After encountering that failed chunk, from a momentary hiccup in the network link, it went ahead and finished the remaining 90 percent over the rest of the night when the network link came back online.

The default download client built into the Web browser can also be used as a download client, using a URL such as https://localhost/DownloadPortal/Download?file=test.txt&chunksize=5242880.

Note that the parameter (chunksize=5242880) is also optional when using the Web browser as a download client. If not included, the server will download the entire file in one chunk using the same HttpResponse.TransmitFile. If included, it will execute a separate HttpResponse.TransmitFile call for each chunk.

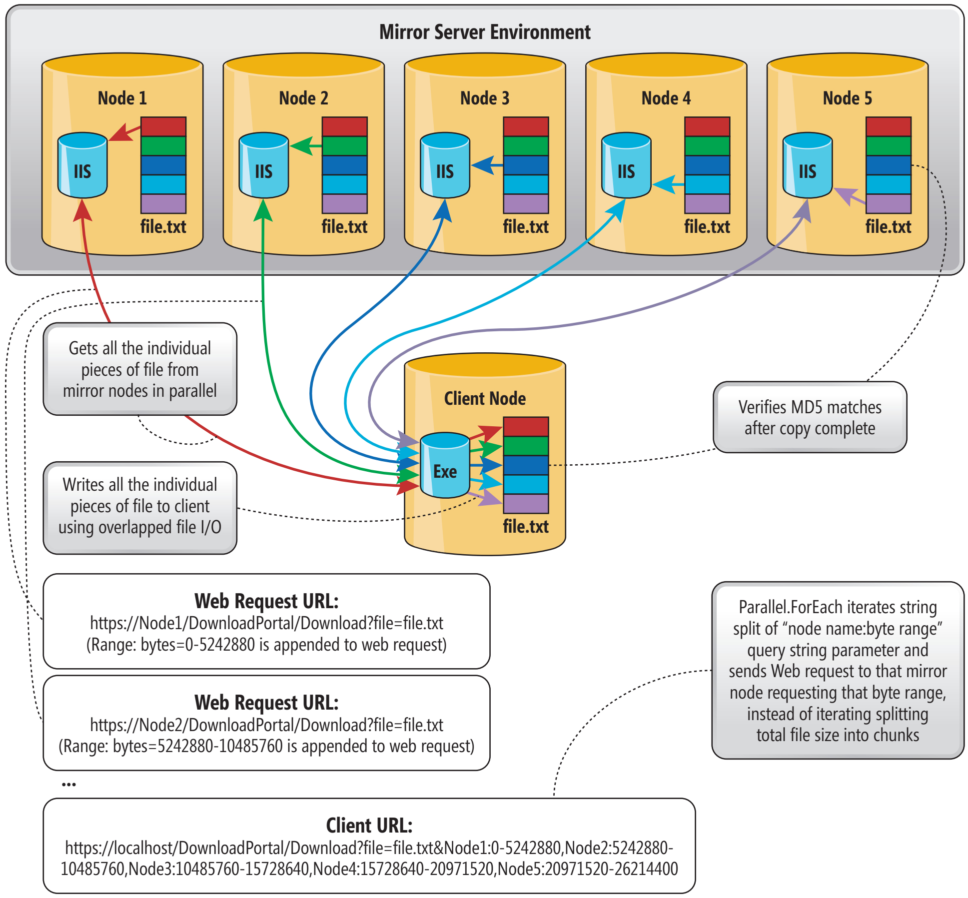


Figure 5 **Hypothetical Future Enhancements to Simulate an Extremely Rudimentary Mirror Infrastructure**

Figure 4 is a high-level overview of the design of DownloadHandler.dll when using a Web browser that doesn't support partial content as a download client. It illustrates the processing flow as the chunks of the file on the server machine's hard drive pass through DownloadHandler.dll and the Web browser into the file on the Web browser machine's hard drive.

A cool feature of our implementation of the IHttpAsyncHandler interface on the IIS Web Server is support of "byte serving" by sending the Accept-Ranges HTTP header in its HTTP response (Accept-Ranges: bytes), telling clients it will serve up portions of a file (partial content range). If the default download client inside the Web browser supports partial content, it can send the server the Range HTTP header in its HTTP request (Range: bytes=5242880-10485760), and when the server sends the partial content back to the client, it will send back the Content-Range HTTP header inside its HTTP response (Content-Range: bytes 5242880-10485760/26214400). So, depending on what Web browser you're using and the default download client built into that browser, you might get some of the same benefits as our standalone executable client. Regardless, most Web browsers will let you build your own custom download client that can be plugged into the browser, replacing the built-in default.

Sample Project Configuration

For the sample project, simply copy DownloadHandler.dll and IOThreads.dll into the \bin directory under the virtual directory and put an entry in the handlers section and modules section of the web.config, like so:

```
<handlers>
  <add name="Download" verb="*" path="Download"
    type="DownloaderHandlers.DownloadHandler" />
</handlers>

<modules>
  <add name="CustomBasicAuthenticationModule" preCondition="managedHandler"
    type="DownloaderHandlers.CustomBasicAuthenticationModule" />
</modules>
```

If there's no virtual directory on the IIS Server, create one with a \bin directory, make it an Application and make sure it's using a Microsoft .NET Framework 4 Application Pool.

The custom basic authentication module uses the same, easy-to-use, AspNetSqlMembershipProvider used on many ASP.NET Web sites today, storing the username and password required to download a file inside the aspnetdb database on the SQL Server. One of the handy benefits of AspNetSqlMembershipProvider is the user needn't have an account on the Windows domain. Detailed instructions

on how to install AspNetSqlMembershipProvider and the settings required on the IIS Server to configure the user accounts and SSL certificate are listed in the sample project code in the comment section at the top of the CustomBasicAuthenticationModule class. The other advanced configuration options used for tuning the IIS Server have usually already been set by the IT department that manages the server and are beyond the scope of this article, but if that's not the case, they're readily available in the TechNet Library at bit.ly/1JRJjNS.

You're done. It's as easy as that.

Compelling Factors

The foremost compelling factor of the design is not that it's faster, but that it's more resilient and fault tolerant to network outages caused by flaky, unstable network links continually going online and offline. Typically, downloading one file, in one chunk, on one connection, will yield the maximum throughput.

There are some unique exceptions to this rule, such as a mirrored server environment where a file is downloaded in separate pieces, getting each piece of the file from a different mirror server, as shown in **Figure 5**. But generally, downloading a file on multiple threads is actually slower than downloading a file on one thread, because the network is typically the bottleneck. However, being able to retry solely the failed portions of the file download repeatedly until they succeed, without having to restart the entire download process, molds what we like to think of as a sort of quasi-fault tolerance.

Also, if someone were to modify the design as a future enhancement to simulate an extremely rudimentary mirror server infrastructure, as shown in **Figure 5**, it might shape what could be thought of as a sort of quasi-redundancy.

Essentially, the design lets you reliably download a file over an unreliable network. A brief hiccup in your network link doesn't mean you have to start over from the beginning; instead, you can simply retry only the pieces of the file that failed. A nice addition to the design (that would make it even more resilient) would be to store the current progress state of the download to a file on the hard drive as the download is progressing, so you could essentially retry a failed download even across client application and client machine restarts. But that will be an exercise left to the reader.

Another compelling factor, which rivals the aforementioned in prominence, lies in the use of HttpResponse.TransmitFile on the server to write the bytes of the file directly to the network stream—with no explicit buffering—in order to minimalize the impact on server memory. It's surprising how negligible the impact is on server memory, even when downloading extremely large files.

There are three additional factors that are far less significant, but compelling nonetheless.

First, because the design includes both the front-end client and the back-end server, you have complete control over the server-side configuration. This gives you the freedom and power to adjust configuration settings that can often greatly impede the file downloading process on servers owned by someone else and out of your control. For example, you can adjust the connection limit restriction imposed per client IP address to a value greater than the usual limit of two connections. You can also adjust the throttling limit per client connection to a greater value.

Second, the sample project code inside our front-end client (FileDownloader.exe) and our back-end server (DownloadHandler.dll) can serve as simple, clear blocks of sample code demonstrating the use of the HTTP request and response headers necessary to facilitate partial content byte ranges in the HTTP protocol. It's easy to see what HTTP request headers the client must send in order to request the byte ranges, and what HTTP response headers the server must send to return the byte ranges as partial content. It should be relatively easy to modify the code to implement higher-level functionality on top of this simple base functionality, or implement some of the more advanced functionality available in more sophisticated software packages. Also, you can use it as a simple starting template that makes it relatively easy to add support for some of the other more advanced HTTP headers, such as Content-Type: multipart/byteranges, Content-MD5: md5-digest, If-Match: entity-tag and so on.

> A brief hiccup in your network link doesn't mean you have to start over from the beginning; instead, you can simply retry only the pieces of the file that failed.

Third, because the design uses the IIS Web Server, you automatically benefit from some of the built-in functionality provided by the server. For example, the communication can automatically be encrypted (using HTTPS with an SSL certificate) and compressed (using gzip compression). However, it might not be advisable to run gzip compression on extremely large files if doing so results in too much stress on your server CPUs. But, in the event your server CPUs can shoulder the additional load, the efficiency of transferring much smaller compressed data can sometimes make a big difference in the overall throughput of the entire system.

Future Improvements

The sample project code only provides the minimum core functionality required for the file downloader to operate. Our goal was to keep the design simple and easy to understand so it could be used relatively effortlessly as a base upon which to add enhancements and additional functionality. It's merely a starting point and base template. Many additional enhancements would be absolutely essential before it could even begin to be used in a production environment. Adding a higher-level abstraction layer that provides this additional, more-advanced functionality is left as an exercise for the reader. However, we'll expound on several of the more crucial enhancements.

The sample project code doesn't currently include an MD5 hash checksum on the file. In the real world, it's essential to employ some sort of file checksum strategy to ensure the file downloaded to the client matches the file on the server, and that it hasn't been tampered

with or altered in any way. The HTTP headers make this easy to do with the header (Content-MD5: md5-digest). In fact, one of our first prototypes included performing an MD5 hash checksum on the file each time the file was requested and placing the digest into the header (Content-MD5: md5-digest) before the file left the server. The client would then perform the same MD5 hash checksum on the file it received and would verify the resulting digest matched the digest in the header (Content-MD5: md5-digest) returned by the server. If it didn't match, the file had been tampered with or corrupted. Although this accomplishes the goal of ensuring the file isn't changed, the large files caused intense CPU pressure on the server and took far too long to perform.

In reality, it will probably require some sort of cache layer that does the MD5 hash checksum processing on the file (in the background) one time for the life of the file, and stores the resulting digest in the dictionary with the file name as the key. Then a simple dictionary lookup is all that's required on the server to obtain the digest for the file, and the digest can be added to the header as the file leaves the server, in a flash, with minimal impact on server CPUs.

We wanted to provide you a simple solid base that includes only the minimum necessary.

The sample project code also doesn't currently restrict a client from using a gigantic number of threads and splitting the file into an enormous number of chunks. It basically allows a client to "do what it needs to do" to ensure it can download a file. In the real world, there would probably need to be some sort of infrastructure that could impose a limit on the client, so one client wouldn't be able to hijack the server and starve all the other clients.

Figure 5 illustrates a hypothetical future enhancement to simulate an extremely rudimentary mirror infrastructure by modifying the design to supplying a list of "node name/byte range" pairs as the URL query string parameter, instead of the current design's "chunksize" parameter. The current design could be modified relatively easily to get each chunk of the file from a different server by merely iterating the "node name/byte range" pairs, launching an HttpWebRequest for each pair, instead of internally iterating to split the total file size into chunks based on the "chunksize" parameter and launching an HttpWebRequest for each chunk.

You could construct the URL for the HttpWebRequest by merely replacing the server name with the associated node name from the list of "node name/byte range" pairs, adding the associated byte range to the Range HTTP header (that is, Range: bytes=0-5242880), and then removing the "node name/byte range" list from the URL entirely. Some sort of metadata file could identify on which servers the pieces of a file are located, and the requesting machine could then assemble the one file from pieces of the file that are spread across different servers.

If a file is mirrored on 10 servers, the design could be modified to get piece 1 of the file from the server 1 mirror copy, piece 2 of the file from the server 2 mirror copy, piece 3 of the file from the server 3 mirror copy and so on. Again, it would be essential to do an MD5 hash checksum on the file after you retrieved all the pieces of the file and reassembled the full file on the client, in order to make sure no chunks were corrupted on any of the mirror servers and that you did in fact receive the entire file. You could even get a little fancier and take it to the next level by having the servers geographically distributed across the country, building some elaborate intelligence into the code that would determine which servers are under the least processing load, then using those servers to service the request returning the chunks of the file.

Wrapping Up

The goal of our design wasn't to create a faster, more scalable file downloader, but to create one that's extremely resilient to momentary network outages.

We took great effort to make sure the design was extremely simple and clearly demonstrated how to use the HTTP protocol headers for "byte serving" byte ranges and partial content.

In our research, we actually found it quite difficult to find a good, clear example of how to do simple HTTP byte serving, and how to properly use the byte range headers in the HTTP protocol. Most examples were either unnecessarily complex or used many of the other headers to implement much more advanced features in the HTTP protocol, making it difficult to understand, let alone try to enhance or expand on going forward.

We wanted to provide you a simple solid base that includes only the minimum necessary, so it would be relatively easy to experiment and incrementally add more advanced functionality over time—or even go so far as to implement an entire higher-level abstraction layer that adds some of the more advanced features of the HTTP protocol.

We simply wanted to provide a straightforward example to learn from and build on going forward. Enjoy! ■

Doug Duerner *is a senior software engineer with more than 15 years designing and implementing large-scale systems with Microsoft technologies. He has worked for several Fortune 500 banking institutions and for a commercial software company that designed and built the large-scale distributed network management system used by the Department of Defense's Defense Information Systems Agency (DISA) for its "Global Information Grid" and the Department of State. He is a geek at heart, focusing on all aspects, but enjoys the most complex and challenging technical hurdles, especially those that everyone says "can't be done." Duerner can be reached at coding.innovation@gmail.com.*

Yeon-Chang Wang *is a senior software engineer with more than 15 years designing and implementing large-scale systems with Microsoft technologies. He, too, has worked for a Fortune 500 banking institution and for a commercial software company that designed and built the large-scale distributed network management system used by the Department of Defense's Defense Information Systems Agency (DISA) for its "Global Information Grid" and the Department of State. He also designed and implemented a large-scale Driver Certification System for one of the world's largest chip manufacturers. Wang has a master's degree in Computer Science. He eats complex problems for dinner and can be reached at yeon_wang@yahoo.com.*

Thanks to the following Microsoft technical experts for reviewing this article: *Stephen Cleary and James McCaffrey*

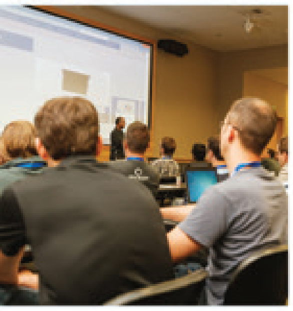
TURN THE PAGE FOR MORE EVENT DETAILS.

Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS

ORLANDO
ROYAL PACIFIC RESORT AT UNIVERSAL ORLANDO

NOV 16-20

Check Out the Additional Sessions for Devs, IT Pros, & DBAs at Live! 360



SQL Server LIVE!
TRAINING FOR DBAs AND IT PROS

SQL Server Live! features 15+ developer sessions, including:

- Encrypting Data in SQL Server - David Dye
- Kick Start! SQL Server 2014 Performance Tips and Tricks - Pinal Dave
- Database Development with SQL Server Data Tools - Leonard Lobel
- Exploring T-SQL Enhancements: Windowing and More - Leonard Lobel
- Python and R for SQL and Business Intelligence Professionals - Jen Stirrup

SharePoint LIVE!
TRAINING FOR COLLABORATION

SharePoint Live! features 12+ developer sessions, including:

- Workshop: Developer OnRamp to the Office 365 & Azure AD Highway! - Andrew Connell
- An Introduction to SharePoint 2013 Add-ins for Developers - Rob Windsor
- SharePoint Developers, Come to the Client Side (We Have Cookies) - Bill Ayers
- SharePoint Development Lifecycle for Solutions and Add-ins - Robert Bogue
- Building Office Add-Ins with Visual Studio - Bill Ayers
- Utilizing jQuery in SharePoint - Get More Done Faster - Mark Rackley
- Using Azure to Replace Server-Side Code in Office 365 - Paul Schaeflein

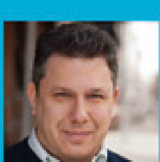
TECHMENTOR
IN-DEPTH TRAINING FOR IT PROS

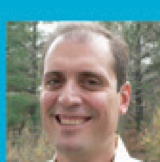
TechMentor features IT Pro and DBA sessions, including:

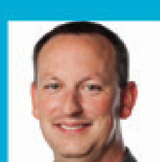
- Workshop: Windows PowerShell Scripting and Toolmaking (BYOL-RA) - Jeffery Hicks
- Your Approach to BYOD Sucks! - Simon May
- Windows Server 2016: Here's What We Know - Don Jones
- Ethical Hacking: Methodologies and Fundamentals - Mike Danseglio & Avril Salter
- DISCUSSION; Career Strategies for the IT Pro - Greg Shields
- PowerShell Unplugged: Stuff I Stole from Snover - Don Jones
- Workshop: SQL Server 2014 for Developers - Leonard Lobel

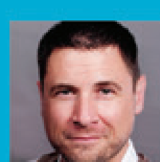
CHECK OUT THE FULL LIVE! 360 AGENDA AT LIVE360EVENTS.COM

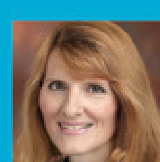
Featured Live! 360 Speakers

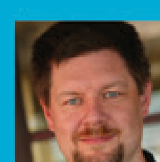

ANDREW BRUST

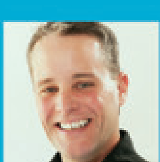

MIGUEL CASTRO

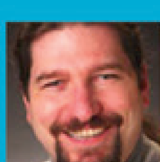

ANDREW CONNELL

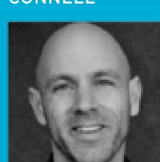

DON JONES


DEBORAH KURATA

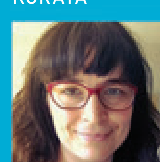

ROCKFORD LHOTKA


MATTHEW MCDERMOTT


TED NEWARD


JOHN PAPA


BRIAN RANDELL


RACHEL REESE


GREG SHIELDS

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

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

START TIME	END TIME					
		Visual Studio Live! & Modern Apps Live! Pre-Conference: Sunday, November 15, 2015				
6:00 PM	9:00 PM	Dine-A-Round Dinner @ Universal CityWalk - 6:00pm - Meet at Conference Registration Desk to walk over with the group				
START TIME	END TIME	**Visual Studio Live! & Modern Apps Live! Pre-Conference Workshops: Monday, November 16, 2015**				
8:00 AM	5:00 PM	**VSM01 Workshop: Service Oriented Technologies: Designing, Developing, & Implementing WCF and the Web API** - *Miguel Castro*	**VSM02 Workshop: Triple D: Design, Development, and DevOps** - *Billy Hollis and Brian Randell*		**VSM03 Workshop: Busy Developer's Guide to MEANJS** - *Ted Neward*	**MAM01 Workshop: Modern App Technology Overview - Android, iOS, Cloud, and Mobile Web** - *Nick Landry, Kevin Ford, & Steve Hughes*
5:00 PM	6:00 PM	EXPO Preview				
6:00 PM	7:00 PM	**Live! 360 Keynote:** Microsoft 3.0: New Strategy, New Relevance - *Pacifica 6* *Mary Jo Foley, Journalist and Author; with Andrew Brust, Senior Director, Datameer*				
START TIME	END TIME	**Visual Studio Live! & Modern Apps Live! Day 1: Tuesday, November 17, 2015**				
8:00 AM	9:00 AM	**Visual Studio Live! & Modern Apps Live! Keynote:** The Future of Application Development - Visual Studio 2015 and .NET 2015 *Jay Schmelzer, Director of Program Management, Visual Studio Team, Microsoft*				
9:00 AM	9:30 AM	Networking Break • Visit the EXPO - *Pacifica 7*				
9:30 AM	10:45 AM	**VST01 AngularJS 101** - *Deborah Kurata*	**VST02 A Tour of Azure for Developers** - *Adam Tuliper*	**VST03 Busy Developer's Guide to NoSQL** - *Ted Neward*	**VST04 Visual Studio, TFS, and VSO in 2015 - What's New?** -*Brian Randell*	**MAT01 Defining Modern App Development** - *Rockford Lhotka*
11:00 AM	12:15 PM	**VST05 From ASP.NET Site to Mobile App in About an Hour** - *Ryan J. Salva*	**VST06 Introduction to Next Generation of Azure PaaS – Service Fabric and Containers** - *Vishwas Lele*	**VST07 Real World SQL Server Data Tools (SSDT)** - *Benjamin Day*	**VST08 Automate Your Builds with Visual Studio Online or Team Foundation Server** - *Tiago Pascoal*	**MAT02 Modern App Architecture** - *Brent Edwards*
12:15 PM	2:00 PM	Lunch • Visit the EXPO - *Oceana Ballroom / Pacifica 7*				
2:00 PM	3:15 PM	**VST09 I Just Met You, and "This" is Crazy, But Here's My NaN, So Call(Me), Maybe?** - *Rachel Appel*	**VST10 Cloud or Not, 10 Reasons Why You Must Know "Websites"** - *Vishwas Lele*	**VST11 Windows 10 for Developers: What's New in Universal Apps** - *Nick Landry*	**VST12 Defensive Coding Techniques in C#** - *Deborah Kurata*	**MAT03 ALM with Visual Studio Online (TFS) and Git** - *Brian Randell*
3:15 PM	4:15 PM	Networking Break • Visit the EXPO - *Pacifica 7*				
4:15 PM	5:30 PM	**VST13 Better Unit Tests through Design Patterns for ASP MVC, WebAPI, and AngularJS** - *Benjamin Day*	**VST14 Running ASP.NET Cross Platform with Docker** - *Adam Tuliper*	**VST15 Build Your First Mobile App in 1 Hour with Microsoft App Studio** - *Nick Landry*	**VST16 Recruiters: The Good, The Bad, & The Ugly** - *Miguel Castro*	**MAT04 Reusing Logic Across Platforms** - *Kevin Ford*
5:30 PM	7:30 PM	Exhibitor Reception				
START TIME	END TIME	**Visual Studio Live! & Modern Apps Live! Day 2: Wednesday, November 18, 2015**				
8:00 AM	9:00 AM	**Live! 360 Keynote:** DevOps: What it Means to You - *Pacifica 6 - Sponsored By* PLURALSIGHT *Don Jones, Curriculum Director for IT Pro Content, Pluralsight & Brian Randell, Partner, MCW Technologies*				
9:15 AM	10:30 AM	**VSW01 Mobile App Development with Xamarin and F#** - *Rachel Reese*	**VSW02 Notify Your Millions of Users with Notification Hubs** - *Matt Milner*	**VSW03 Let's Write a Windows 10 App: A Basic Introduction to Universal Apps** - *Billy Hollis*	**VSW04 To Git or Not to Git for Enterprise Development** - *Benjamin Day*	**MAW01 Coding for Quality and Maintainability** - *Jason Bock*
10:30 AM	11:00 AM	Networking Break • Visit the EXPO - Pacifica 7				
11:00 AM	12:15 PM	**VSW05 Automated UI Testing for Android and iOS Mobile Apps** - *James Montemagno*	**VSW06 Busy Developer's Guide to the Clouds** - *Ted Neward*	**VSW07 Designing and Building UX for Finding and Visualizing Data in XAML Applications** - *Billy Hollis*	**VSW08 Anything C# Can Do, F# Can Do Better** - *Rachel Appel & Rachel Reese*	**MAW02 Start Thinking Like a Designer** - *Anthony Handley*
12:15 PM	1:45 PM	Birds-of-a-Feather Lunch • Visit the EXPO - *Oceana Ballroom / Pacifica 7*				
1:45 PM	3:00 PM	**VSW09 Stop Creating Forms In Triplicate - Use Xamarin Forms** - *Matt Milner*	**VSW10 Build Universal HTML5 Web Apps Within Visual Studio** - *Sandeep Adwankar, Product Manager*	**VSW11 Developing Awesome 3D Games with Unity and C#** - *Adam Tuliper*	**VSW12 Unit Testing Makes Me Faster: Convincing Your Boss, Your Co-Workers, and Yourself** - *Jeremy Clark*	**MAW03 Applied UX: iOS, Android, Windows** - *Anthony Handley*
3:00 PM	4:00 PM	Networking Break • Visit the EXPO • Expo Raffle @ 3:30 p.m. - *Pacifica 7*				
4:00 PM	5:15 PM	**VSW13 Go Mobile with C#, Visual Studio, and Xamarin** - *James Montemagno*	**VSW14 To Be Announced**	**VSW15 Automated Cross Browser Testing of Your Web Applications with Visual Studio CodedUI** - *Marcel de Vries*	**VSW16 DI Why? Getting a Grip on Dependency Injection** - *Jeremy Clark*	**MAW04 Leveraging Azure Services** - *Kevin Ford*
8:00 PM	10:00 PM	Live! 360 Dessert Luau - *Wantilan Pavilion - Sponsored by* amazon web services				
START TIME	END TIME	**Visual Studio Live! & Modern Apps Live! Day 3: Thursday, November 19, 2015**				
8:00 AM	9:15 AM	**VSH01 Getting Started with ASP.NET 5** - *Scott Allen*	**VSH02 Lessons Learned: Being Agile in a Waterfall World** - *Philip Japikse*	**VSH03 Windows, NUI and You** - *Brian Randell*	**VSH04 Improving Performance in .NET Applications** - *Jason Bock*	**MAH01 Building for the Modern Web with JavaScript Applications** - *Allen Conway*
9:30 AM	10:45 AM	**VSH05 Build Data Driven Web Applications with ASP.NET MVC** - *Rachel Appel*	**VSH06 User Story Mapping** - *Philip Japikse*	**VSH07 Building Adaptive Uis for All Types of Windows** - *Ben Dewey*	**VSH08 Asynchronous Tips and Tricks** - *Jason Bock*	**MAH02 Building a Modern App with Xamarin** - *Nick Landry*
11:00 AM	12:15 PM	**VSH09 Automated Cross Browser Testing of Your Web Applications with Visual Studio CodedUI** - *Marcel de Vries*	**VSH10 Performance and Debugging with the Diagnostic Hub in Visual Studio** - *Sasha Goldshtein*	**VSH11 XAML Antipatterns** - *Ben Dewey*	**VSH12 Roslyn and .NET Code Gems** - *Scott Allen*	**MAH03 Building a Modern Cross-Platform App** - *Brent Edwards*
12:15 PM	1:30 PM	Lunch on the Lanai - *Lanai / Pacifica 7*				
1:30 PM	2:45 PM	**VSH13 Hate JavaScript? Try TypeScript.** - *Ben Hoelting*	**VSH14 Advanced Modern App Architecture Concepts** - *Marcel de Vries*	**VSH15 WPF MVVM In Depth** - *Brian Noyes*	**VSH16 Getting More Out of Visual Studio Online: Integration and Extensibility** - *Tiago Pascoal*	**MAH04 DevOps And Modern Applications** - *Dan Nordquist*
3:00 PM	4:15 PM	**VSH17 Grunt, Gulp, Yeoman and Other Tools for Modern Web Development** - *Ben Hoelting*	**VSH18 The Vector in Your CPU: Exploiting SIMD for Superscalar Performance** - *Sasha Goldshtein*	**VSH19 Building Maintainable and Extensible MVVM WPF Apps with Prism** - *Brian Noyes*	**VSH20 Readable Code** - *John Papa*	**MAH05 Analyzing Results with Power BI** - *Steve Hughes*
4:30 PM	5:45 PM	Live! 360 Conference Wrap-Up - *Andrew Brust (Moderator), Andrew Connell, Don Jones, Rockford Lhotka, Matthew McDermott, Brian Randell, & Greg Shields*				
START TIME	END TIME	**Visual Studio Live! & Modern Apps Live! Post-Conference Workshops: Friday, November 20, 2015**				
8:00 AM	5:00 PM	**VSF01 Workshop: Angular in 0 to 60** - *John Papa*		**VSF02 Workshop: Native Mobile App Development for iOS, Android and Windows Using C#** - *Marcel de Vries & Roy Cornelissen*		**MAF01 Workshop: Modern App Development In-Depth - iOS, Android, Windows, and Web** - *Brent Edwards, Anthony Handley, & Allen Conway*

Sessions and speakers subject to change.

PRODUCED BY
1105 MEDIA
YOUR GROWTH. OUR BUSINESS.

Manage Technical Debt with SonarQube and TFS

Cesar Solis Brito and Hosam Kamel

Technical debt is the set of problems in a development effort that make progress on customer value inefficient. It undermines productivity by making code hard to understand, fragile, time-consuming to change, difficult to validate and creates unplanned work that blocks progress. Technical debt saps an organization's strength because of high customer-support costs and, eventually, some combination of these issues produces larger problems.

Technical debt is insidious. It starts small and grows over time through rushed changes and lack of context and discipline. It can materialize out of nowhere—even for a project regarded as "clean"—because of changes in project circumstances. For example, prototype code produced for the U.S. market might be proposed for international, instantly creating debt related to localizability. Or, as technologies evolve, the app might not keep up.

This article discusses:

- The different forms of technical debt and the core problems it causes for dev teams
- Setting up the Visual Studio Team Foundation Server and SonarSource SonarQube environment
- Microsoft Azure Infrastructure as a Service
- Strategy to manage technical debt based on the SQALE method

Technologies discussed:

SonarQube, Visual Studio Team Foundation Server

Debt comes in many forms, including problems found through code analysis, duplicate code, code complexity, not enough tests, overlapping and flaky tests, and "architectural spaghetti."

> Technical debt has to be managed—not necessarily eliminated—and, thus, has to be acknowledged and measured.

The core problems faced by development teams in managing their technical debt include:

- Understanding and identifying the debt and where it's located in the code.
- Evaluating the cost of remediation and the cost of non-remediation. Fixing technical debt has a cost. Not fixing it also has a cost, which is even more complex to evaluate and can be bigger.
- Putting policies in place focused on preventing debt from getting worse and managing it down.
- Exposing developers to the debt required to meet the policies in a way that isn't overwhelming, so that they can tackle it as a natural part of their day-to-day development process and don't regard it as a huge time-consuming and tiresome chore.

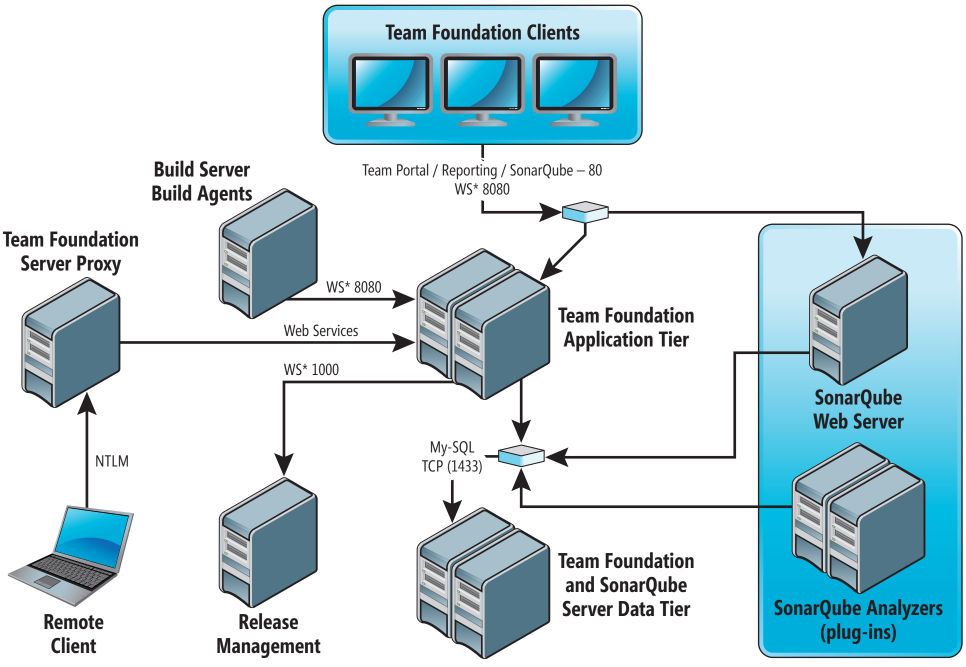


Figure 1 **Team Foundation Server Reference Architecture, Including SonarQube**

- Tracking debt over time to ensure that it's trending in the right direction and meeting defined policies.
- Remediating debt as efficiently and automatically as possible.

So, this debt has to be managed—not necessarily eliminated—and, thus, has to be measured. When managed down, it should be prioritized among other new features and desired functionality. The impact should be measured, as well, just as the financial impact of acquiring a home loan is measured.

You need to be conscious of the decisions you make that might increase technical debt during development. You should be able to measure the improvements achieved by actually managing technical debt. This is important to justify the product owners' (clients, users, managers and so on) investment.

Solutions such as SonarQube from SonarSource collaborate with Visual Studio Team Foundation Server (TFS) to provide strategies to facilitate data gathering and present it in a way that helps manage and reduce technical debt. SonarQube is an open source platform for understanding and managing technical debt.

In this article, we'll focus on a method of measuring and actively managing technical debt. Called Software Quality Assessment based on Lifecycle Expectations (SQALE), it's an open source method that focuses on non-functional quality attributes of a software code base (bit.ly/1JQ96qT). The SQALE method was developed by Inspearit France (formerly DNV ITGS France).

According to a SonarSource blog post, technical debt is like a home water leak (bit.ly/1NOY4A9). When you have a water leak at home, what do you do first, plug the leak or mop the floor? The answer is simple and intuitive: You plug the leak. Why? Because any other action will be useless; it will be only a matter of time before the same amount of water is back on the floor.

With technical debt, "fixing the leak" means putting the focus on the "new" code; that is, the code that was added or changed since the last release, or the last commit/check-in.

Implementing SonarQube and TFS

Figure 1 shows a reference architecture for an enterprise setup of TFS and SonarQube; there's detailed guidance on capacity, design, implementation and operation of TFS on-premises and in Microsoft Azure Infrastructure as a Service (IaaS) in the "TFS Planning Guide" and "TFS on Azure IaaS Guide," which can be downloaded from the Visual Studio ALM Rangers CodePlex site (aka.ms/treasure5). This architecture and its implementation are directed toward enterprise-level support by considering high-availability features in the data tier through a database cluster, and in the application tier with a farm.

A scalable SonarQube server deployment can be achieved by setting up multiple SonarQube servers based on individual teams, or by technology taking into consideration the following:

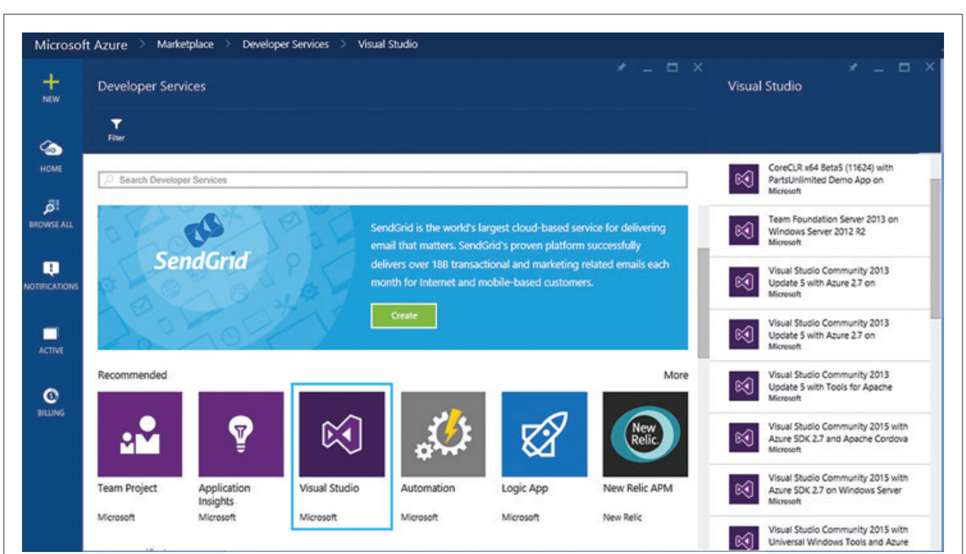


Figure 2 **Virtual Machines Template Available on Microsoft Azure**

A SonarQube database can be installed within an instance in the database cluster providing a secure way to handle high-availability services.

SonarQube analyzer (plug-in) implements a scalable per-server way of processing loads of analysis data into the SonarQube database.

You need to be conscious of the decisions you make that might increase technical debt during development. With technical debt, "fixing the leak" means putting the focus on the "new" code.

The database and the analyzers must be located in the same network. SonarSource has been working on refactoring SonarQube (version 5.2) so that it has a three-tier architecture, which will make this constraint obsolete.

All the machines—database, Web server, analyzers—must be time-synchronized.

Prerequisites for Hosting SonarQube and TFS on Azure

To set up a TFS and SonarQube environment on Azure you need to have an Azure subscription with enough credit toward the size of the environment you're going to create. You can use the TFS Planning Guide (bit.ly/1JzfVJK) to decide on the sizing of Virtual Machines (VMs). The guide also provides recommendations on authentication models such as Stand-alone, Extended Domain and One Way trust options. A great feature of Azure that can help accelerate the environment creation is the available ready-made TFS VM template, as shown in **Figure 2**.

Network configuration should be selected according to your needs for protecting external access to inner layers of the TFS architecture. Various ways of connecting Azure to on-premises through a VPN should be reviewed as a way to extend corporate infrastructure and gain cloud elasticity. The guide describes best practices and performance tips that are helpful in setting up the TFS environment on Azure. SonarQube requires that the database and analyzers must be located in the same network.

Follow these recommended steps when planning for the installation:

- **Determine Azure IaaS suitability.** Cloud computing delivers computing capabilities as a service, giving you access to resources such as compute power, networking and storage. Decide whether the constraints are acceptable and whether there's a value-add.
- **TFS planning.** Use the "TFS Planning and Disaster Avoidance and Recovery Guide" (bit.ly/1JzfVJK) to determine the best server architecture for your requirements.
- **Azure IaaS mapping.** Map to the closest Azure VM configuration based on cost and scalability.
- **Azure platform governance.** Define the account, subscriptions and ownership of administration and reporting to meet your enterprise auditing requirements. Clearly define ownership and responsibilities.
- **Network planning.** Define your network, which encompasses on-premises requirements, network address segments, affinity grouping, name resolution, Azure tunneling and monitoring.
- **Storage planning.** Define your storage strategy using on-premises storage or Azure storage.
- **Validate.** Contact an Azure subject matter expert from Microsoft or Azure communities (bit.ly/1fVTgz0) and validate the plans before committing yourself. Ensure the planned environment is technically and financially viable and can be maintained.

As the intention of this article is to set up a SonarQube server for demo and trial purposes, additional steps are not mentioned here. If you're planning or provisioning a production environment or a migration from an existing one, you should review the other steps in the guide for a complete plan. There's also a deployment checklist in the guide that should be followed for a successful installation.

The Azure IaaS reference architecture and installation exercised here is based on the companion proof of concept (POC) detailed in the "TFS on Azure IaaS Guide," which can be used for sample implementation. While the guide contains detailed steps for setting up the TFS POC environment, there's no constraint around using the same instructions with your own POC environment or the production one.

SonarQube Installation Requirements

We recommend using the same IaaS SQL Server instance for SonarQube installation, just taking care of specific SQL Server configuration requirements when creating the SonarQube database. The "SonarQube Installation Guide for Existing Team Foundation Server 2013 Single Server Environment" (bit.ly/1itxhS9) has plenty of detailed information and references on how to set up Azure Regions (make sure to co-locate the SonarQube server VM in the same region as TFS) and SQL Server. The SQL Server configuration and database creation has specific requirements to meet for SonarQube. For example, the SQL Database should be case-sensitive and accent-sensitive and this is not the default. SQL authentication also needs to be enabled and a SQL user configured for SonarQube to work. When hosting on Azure you need to change network protocols for SQL Server to enable access to the

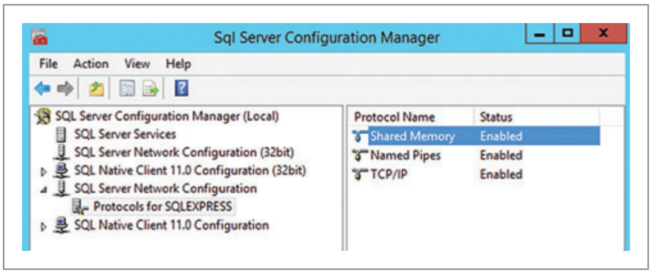


Figure 3 **Enabling the Named Pipes and the TCP/IP Protocol for the SQL Express Database**

database from outside the VM by following certain steps. First, using the SQL Server Configuration manager, enable the Named Pipes and the TCP/IP protocol for the SQL Express database, as shown in **Figure 3**.

Still in the Server Configuration manager, edit the TCP/IP properties, and in the IP Addresses tab, look for TCP Dynamic Ports and make a note of the port (here 49419), as shown in **Figure 4**. You'll use it to enable the connection to the database from the outside.

As shown in **Figure 5**, restart the SQL Server by right-clicking on SQL Server Express in the SQL Server Services item and choose Restart. Start the SQL Server Browser (in the Service Tab change it from Stopped to Automatic and then start it).

You now need to open a port in the firewall so that the database is accessible from the TFS machine. Type the following in the cmd.exe prompt running as administrator:

```
netsh advfirewall firewall add rule name="SQL" dir=in action=allow
  protocol=TCP localport=49419 49590
```

SonarQube doesn't require the full JVM installation—Java SE Runtime Environment (JRE) should be enough and there are specific recommendations on how to set it up and take advantage of a x64 processor architecture.

The Setup of the SonarQube Server section in the guide describes the download, installation and configuration procedures when configuring and starting SonarQube, including Windows Firewall settings.

When hosting on Azure you need to enable access to the SonarQube server to be accessible from the outside world. To do so, first turn off anonymous access. To force user authentication, log in as a system administrator and go to Settings | General Settings | Security and set the Force user authentication property to true.

Open the port in the firewall for the Web server. Now you need to open a port in the firewall so that the Web server is accessible from the Internet. In a cmd.exe prompt running as administrator, type:

```
netsh advfirewall firewall add rule name="Sonar Web" dir=in action=allow
  protocol=TCP localport=9000
```

Add EndPoints in Azure for Sonar Web and the database. For the moment, you can log in to the SonarQube machine with a remote desktop and access the database and the Web site. You need to ensure you can connect to these two resources from your machines over the Internet. For this, you need to create Endpoints in Azure:

- Go to portal.azure.com.
- Browse to find the virtual machine where SonarQube is set up.
- In the blade presenting this VM, click on All Settings.
- In the settings, click Endpoints.
- In the Endpoints blade, click Add.
- Add an endpoint named Sonar Web mapping the TCP public port 9000 to the private port 9000.
- Add an endpoint named SQL mapping the TCP public port 1433 to the private port 49419 (the port 1433 is expected by the SQL Server Management Studio or Visual Studio to discover SQL-EXPRESS). Note that this step won't be necessary any longer when SonarQube 5.2 is released.

SonarQube can perform analysis on many languages; first you need to install the plug-ins for the languages of the project to be analyzed from the SonarQube plug-in library (bit.ly/1NdhnYF) or through SonarQube Update Center (bit.ly/1NPKZet). Then you need to install and configure some prerequisites on the build agent, depending on the type of projects: MSBuild.SonarQube.Runner (bit.ly/1LOYzM3), a for .NET- based projects); SonarQube Maven Plugin runner (for Java Maven projects); and SonarQube Runner, a generic runner for all other projects types.

Integrate with Team Build and test the modified build definition, as shown in **Figure 6**. The SonarQube Runner tool integrates well with Build Agent so it can easily be taken over by development teams used to work with TFS.

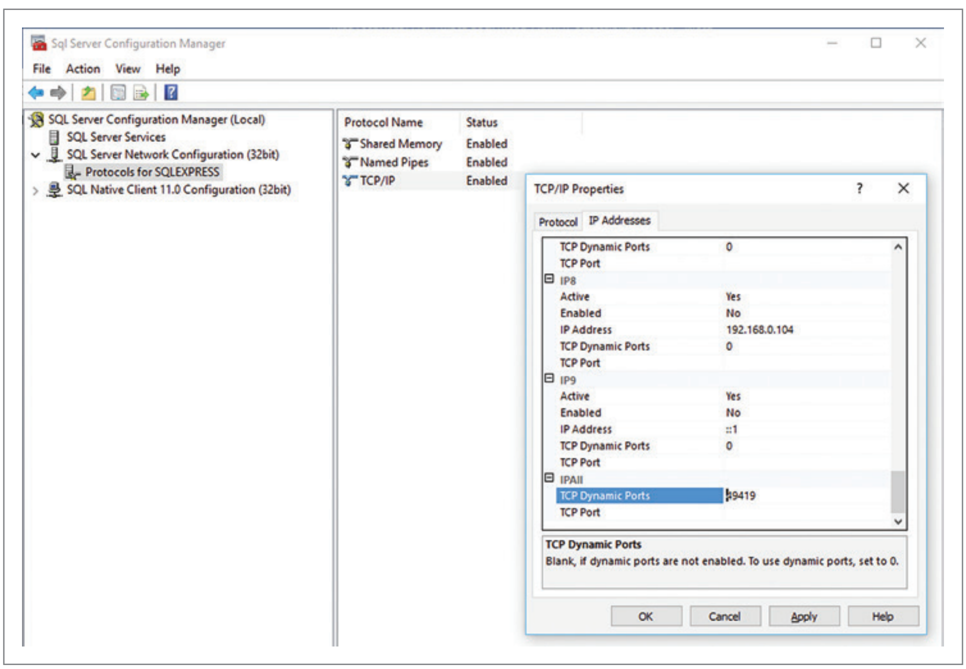


Figure 4 **Note the Port Number in TCP Dynamic Ports to Enable the Connection to the Database from the Outside**

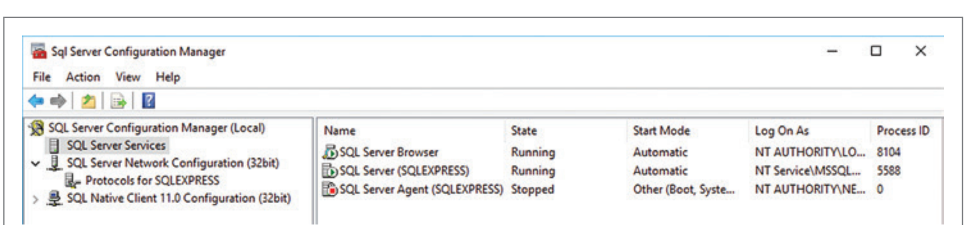


Figure 5 **Start the SQL Server Browser**

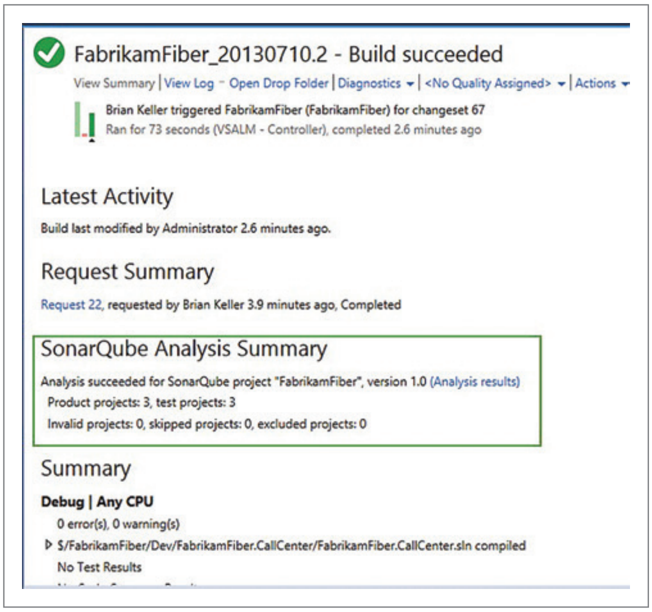


Figure 6 **SonarQube Performing Analysis with Build Agent**

Strategy to Manage Technical Debt

Now that the SonarQube environment is set up, we'll describe a strategy to leverage technical debt measurement within the agile virtuous cycle to improve the value to the product owner and the business. As noted by the SQALE method, our intention is to define clearly what creates the technical debt, estimate correctly this debt, analyze the debt upon technical and business perspective, and offer different prioritization strategies allowing establishing an optimized payback plan.

The SQALE method, as shown in **Figure 7**, is used for formulating and organizing the non-functional requirements that relate to the code's quality. It's organized in three hierarchical levels: characteristics, sub-characteristics and requirements related to the code's internal attributes. (These requirements usually depend on the software's context and language. Any violation of these requirements introduces technical debt.)

The SQALE method normalizes the reports resulting from the source code analysis tools by transforming them into remediation costs and non-remediation costs. It also defines rules for aggregating these costs, either following the SQALE method's tree structure, or following the hierarchy of the source code's artifacts. In order to manage the technical debt of a software project, you should make the debt visible. Make sure the product owner and marketing personnel know that technical debt exists and repeat to them often, "If we don't schedule time to pay off technical debt, you may not get all of the features you want."

The water leak metaphor recommends postponing the current technical debt, except what's really important and urgent to fix—for instance, security issues. Thus, take a baseline, make sure that you're aware of any new debt, try not to introduce any new debt (stop the leak) and as you're working and refactoring code, take the time to fix the existing debt. You're unit testing it, so this isn't dangerous to do so (clean up as you go).

Managing technical debt doesn't need to be budgeted. It's a change of behavior in the entire organization.

As seen in **Figure 8**, SonarQube presents a dashboard based on SQALE in order for the team to have a common representation of the quality state of a software project.

Figure 9 shows the Technical Debt Pyramid widget. Read the graph vertically from bottom to top. First, you want to ensure you're testable (40min), otherwise, when you change code, there's no guarantee you won't introduce more problems. Then you want to be reliable, which will take an extra 42min; so in total, to be reliable you need to spend 1h 22min and so on.

How to Manage the Technical Debt?

Following are the main steps required to manage the technical debt:

Step 1: Set project goals. There should be a definition of what creates technical debt. The indices are computed based on the average remediation efforts estimated by the development team. Several graphs and diagrams are used to efficiently visualize the strengths and weaknesses of the assessed source code. You should ask:

- Which software pieces contribute the most to the risks?
- Which quality criteria are the most impacted?
- What is the impact of the identified non-compliances on the project or its users?

These are defined goals over a project. SonarQube allows you to drill down to specific issues within the code in order to understand the reason and implications and, thus, plan tasks for paying back the debt. This analysis should be made during agile planning and, if possible, prioritized along Product Backlog Items (PBIs). The PBIs are defined and evaluated to balance or improve velocity or

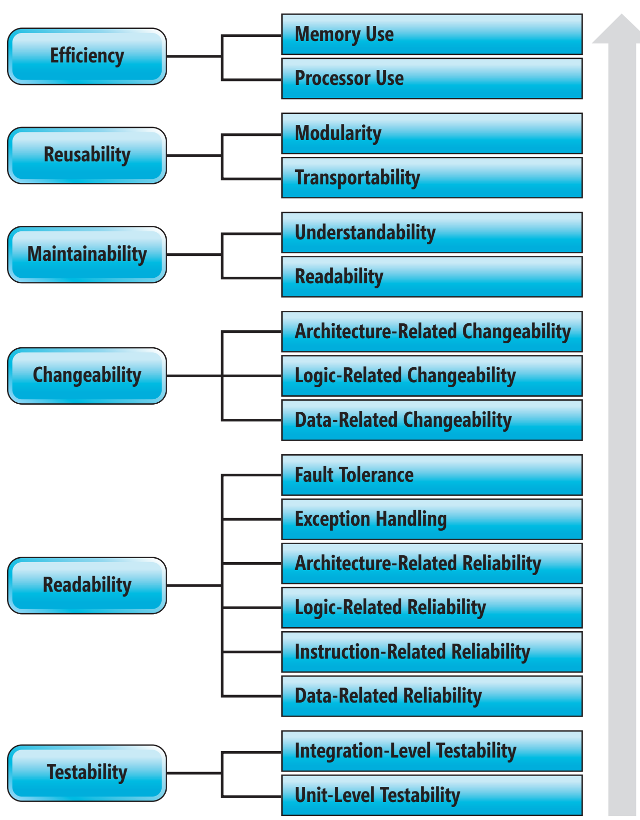


Figure 7 **The SQALE Method**

Work With Relational Data, Not Complex APIs or Services

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

Give RSSBus a try today and see what mean:

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

rssbus

INTEGRATION YOUR WAY

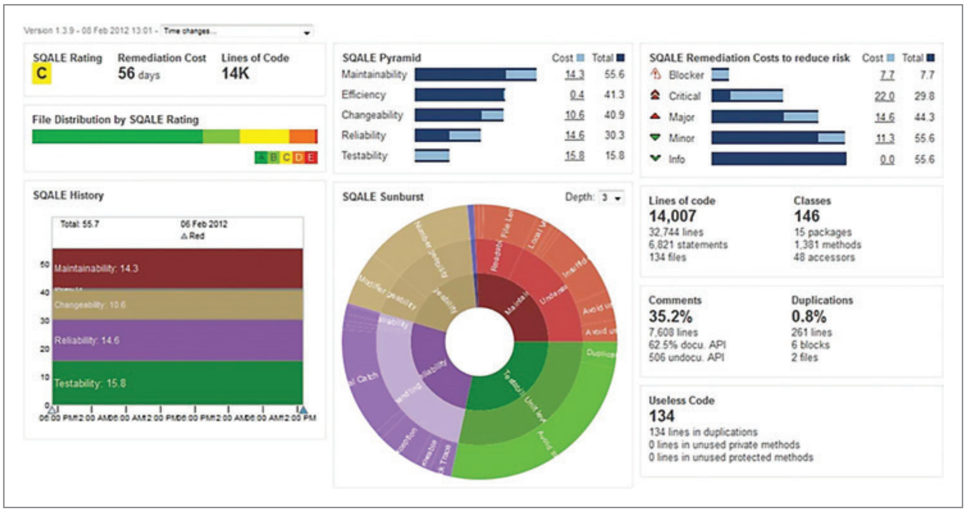


Figure 8 **The SQALE Method-Based SonarQube Dashboard Within a Project Lifetime**

other architecture characteristics. The characteristics are defined from more critical, to the ones that will deliver potential business value, to the product users like portability or reusability.

Step 2: Monitor the amount of technical debt over time. Be able to check results of tasks executed or best practices being implemented. Questions to ask include whether the technical debt increased during the last day/sprint/version and how much margin there is related to the goal set for the project.

Step 3: Analyze the process and the impact. You should also be able to compare technical debt by following up practices between different projects or teams or subcontractors. This could improve practices between teams while taking best practices and applying them more effectively while also better measuring technical debt results.

It's possible to analyze the origin of the technical debt by answering questions such as Which part of the current debt has been created during the last day/sprint/version, which part of the current debt is inherited from legacy code and which parts of the project have the highest technical debt.

Teams should also focus on specific parts of technical debt in order to optimize the results. For this, ask questions such as which are the most urgent issues to fix within my code, what are the next issues to fix and which violations of "right code" are not so costly to fix and will generate a high decrease of the impact for business.

Finally, if I spend 100 hours to decrease the technical debt, ask how much I will improve my velocity? And what improvement will users perceive on the quality of the application?

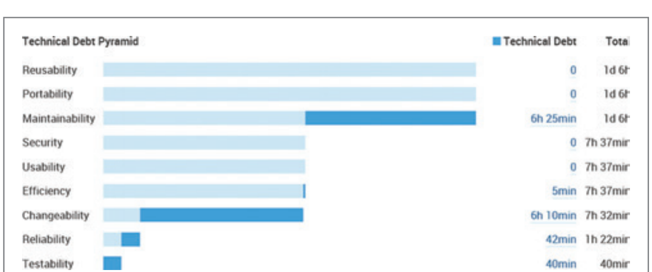


Figure 9 **The Technical Debt Pyramid Widget**

The answers to these questions will provide a natural feedback mechanism for finding opportunities for improvement and closing the gaps to a better product.

The SQALE method supports three strategies for addressing technical debt:

1. Follow the technical logic (avoid useless rework). Use the SQALE pyramid.
2. Decrease the business impact by fixing the non-conformities with the highest business impact.
3. Optimize your ROI by fixing the non-conformities with the highest ROI.

Wrapping Up

Technical debt has to be managed—not necessarily eliminated—and, thus, has to be acknowledged and measured. Choosing to stop it is a good start. You might choose to manage it down, and there are different ways. You might clean up existing debt as you touch code to fix bugs or add new features. Or you might decide remediation actions are important to take for security or conformance issues. Perhaps this is the way your organization thinks it should address the problem. In that case, the work needs to be prioritized among the other new features or functionality desired, measuring the impact on the implications.

You should learn to be conscious of the decisions you make that might increase technical debt and you should be able to measure the improvements achieved by actually managing it.

SonarQube and the TFS integration components help facilitate gathering data from build in the Microsoft Build Engine, or TFS and Visual Studio Online. SonarQube presents technical debt in a way that helps you understand it and plan how to better invest in resolving it. There is ongoing work done by Microsoft and SonarSource (see bit.ly/10eqftX) to improve this integration further and provide a first-class solution for managing technical debt on a platform. ■

Cesar Solis Brito *has been a consultant specializing in software development in the financial and pharmaceutical fields and at Microsoft for more than 20 years. Currently, he's an escalation engineer at Microsoft for the Azure Platform. He has been actively working in ALM and related Microsoft development technologies. You can reach him at Cesar.Solis@microsoft.com or follow him on Twitter: @cesarsolis.*

Hosam Kamel *is a senior premier field engineer at Microsoft, and a Visual Studio ALM Ranger with more than 10 years of software development experience. He currently specializes in Visual Studio ALM and Team Foundation Server. You can reach him via his blog at blogs.msdn.com/HKamel or follow him on Twitter: @HosamKamel.*

Thanks to the following Microsoft technical experts for reviewing this article: *Mathew Aniyan, Brian Blackman, Harysh Menon, Duncan Pocklington and Jean-Marc Prieur* msdn magazine VP

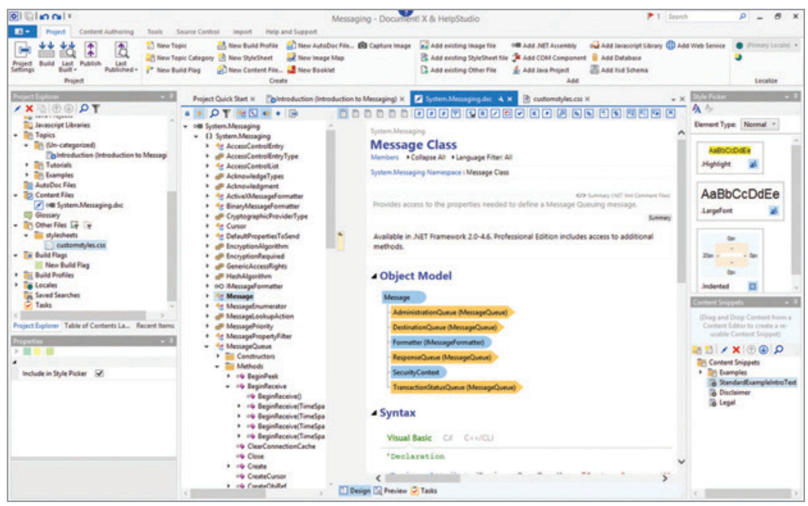
MSDN MAGAZINE VENDOR PROFILE

Innovasys

Documentation and Help Authoring Solutions from Innovasys

Q&A with Richard Sloggett, Chief Technical Officer at Innovasys

Innovasys has been a leading provider of Documentation and Help Authoring tools since 1998 and is focused on producing tools that enable developers and technical writers worldwide to produce professional quality documentation, help systems and procedures with minimum friction.



Q What is Document! X?
A Document! X is a documentation and help authoring solution that combines the ability to automatically generate documentation with a full featured and mature content authoring environment. This unique combination delivers all the time and cost benefits of automating as much of the documentation workflow as possible whilst retaining the ability to fully customize and supplement the automatically generated output.

Q What kinds of documentation can Document! X help produce?
A Document! X can document .NET assemblies, Web Services (REST or SOAP), Databases (Access, SQL Server, Oracle, OLE DB), XSD Schemas, JavaScript, COM Components, Type Libraries and Java. In addition to supplementing the automatically generated content, you can also author any number of free format conceptual Topics to round out your documentation.

The same rich and mature authoring environment and fully customizable templates are available for all of the supported documentation types.

Q How does Document! X make documentation less painful?
A With Document! X, documentation can be automatically produced throughout design, development and beyond without requiring investment of developer resources, providing development teams with an accurate and up to date reference and allowing new developers to jump the learning curve of new components, schemas or web services. Document! X makes producing documentation a natural and productive activity for developers and technical writers alike.

Q Can Document! X edit Xml comments in Visual Studio?
A The Document! X Visual Comment editor integrated with Visual Studio provides a WYSIWYG environment in which you can create and edit your Xml source code comments. Content can be authored both in source code comments and in Document! X Content Files outside of the source. This approach makes it easy to make post-development edits and combine the efforts of developers and technical writers.

Q Why choose Document! X?
A Document! X has been trusted by developers and technical writers worldwide since 1998 to produce accurate and comprehensive professional quality documentation. We believe that Document! X is the most comprehensive and mature documentation tool available today, backed with first class support, and we encourage you to try our free trial version to document your own components and see if you agree.

For more information and a free trial version, visit → www.innovasys.com

JAMES MCCAFFREY

The T-Test Using C#

The t-test is one of the most fundamental forms of statistical analysis. Its goal is to determine whether the means (averages) of two sets of numbers are equal when you only have samples of the two sets. The idea is best explained by example. Suppose you're investigating the mathematical ability of high school males and females in a large school district. The ability test is expensive and time-consuming so you can't give the test to all the students. Instead, you randomly select a sample of 10 males and 10 females and give them the math test. From the sample results you can perform a t-test to infer whether the true average score of all the males is equal to the true average score of all the females.

There are many standalone tools, including Excel, which can perform a t-test. But if you want to integrate t-test functionality directly into a software system, using standalone tools can be awkward or impossible, and may involve copyright or other legal issues. This article explains how to perform a t-test using raw (no external libraries) C# code.

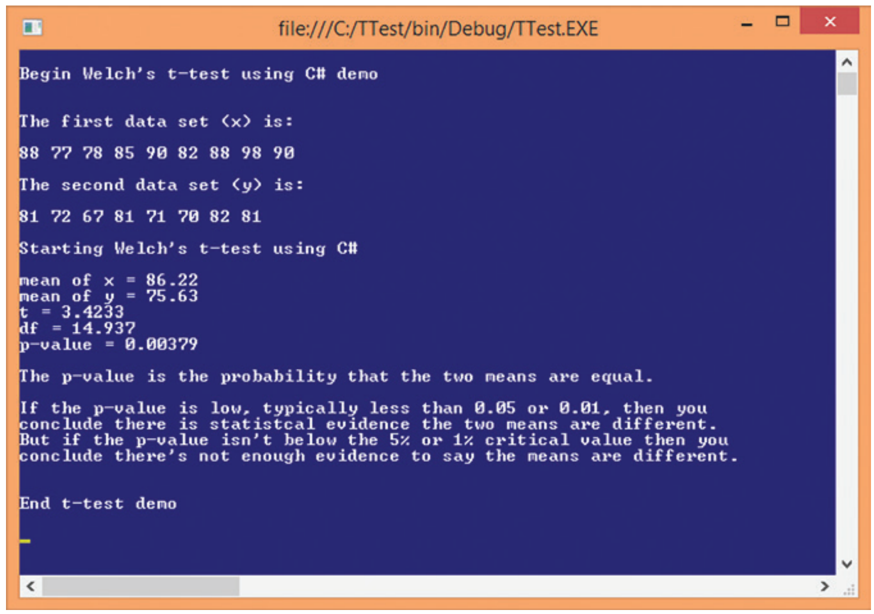


Figure 1 **Demo of the T-Test Using C#**

The best way to get a feel for what the t-test is and to see where this article is headed is to take a look at the demo program in **Figure 1**. The first data set is { 88, 77, 78, 85, 90, 82, 88, 98, 90 }. You can imagine these are the test scores of 10 males, where one of the males dropped out for some reason, leaving just nine scores.

The t-test is one of the most fundamental forms of statistical analysis.

The second data set is { 81, 72, 67, 81, 71, 70, 82, 81 }. You can imagine these are the test scores of 10 females, where two of the females dropped out for some reason, leaving just eight scores. The mean of the first data set is 86.22 and the mean of the second data set is 75.63, which suggests that the means of the two groups are not the same because there's an almost 11-point difference. But even if the overall average scores of the two groups (all males and all females) were in fact the same, because only samples are being used, the difference in the sample averages could have happened by chance.

Using the two sample data sets, the demo program calculates a "t-statistic" (t) with value 3.4233 and a "degrees of freedom" (often abbreviated as df, or indicated by the lowercase Greek letter nu, ν) value of 14.937. Then, using the t and df values, a probability value (p-value) is calculated, with value 0.00379. There are several forms of the t-test. Perhaps the most common is called the Student t-test. The demo uses an improved variation called the Welch t-test.

The p-value is the probability that the true averages of the two populations (all males and females) are actually the same, given the sample scores and, therefore, that the observed difference of about 11 points was due to chance. In this case, the p-value is very small so you'd conclude that the true averages of all males and all females are *not* equal. In most problems, the critical p-value for comparison with the calculated p-value is arbitrarily defined to be 0.01 or 0.05.

Put somewhat differently, if the true average scores for all males and females were the same, the probability that you'd see the observed difference of nearly 11 points in the two sample averages of size nine and eight is only 0.00379—extremely unlikely.

This article assumes you have at least intermediate programming skills but doesn't assume you know anything about the t-test. The demo is coded using C#, but you shouldn't have much trouble if

Code download available at msdn.com/magazine/msdnmag1115.

you want to refactor the code to another language, such as Visual Basic .NET or JavaScript.

Understanding the T-Distribution

The t-test is based on the t-distribution. And the t-distribution is closely related to the normal (also called Gaussian, or bell-shaped) distribution. The shape of a normal distribution set of data depends on both the mean and the standard deviation of the data. The standard deviation is a value that measures how spread out, or variable, the data is. A special case is when the mean (often indicated by Greek letter mu, μ) is 0 and the standard deviation (often abbreviated in English as sd, or indicated by Greek letter sigma, σ, is 1. The normal distribution with mean = 0 and sd = 1 is called the standard normal distribution. Its graph is shown in **Figure 2**.

In **Figure 2**, the equation that defines the standard normal distribution is called the probability density function. The t-distribution closely resembles the normal distribution. The shape of a t-distribution depends on a single value called the "degrees of freedom." The t-distribution with df = 5 is shown in **Figure 3**.

In **Figure 3**, the equation that defines the t-distribution involves the Gamma function, which is indicated by the Greek capital letter gamma (Γ). In order to perform a t-test, you need to calculate and sum two identical areas under the curve of the t-distribution. This combined area is the p-value. For example, in **Figure 3**, if the value of t is 2.0, the combined areas under the curve you need are from -infinity to -2.0, and +2.0 to +infinity. In this case the combined area, which is the p-value, is 0.101939. For the demo program, when t = 3.4233, the combined area is 0.00379.

OK, but how can the area under the t-distribution be calculated? There are several approaches to this problem, but the most common technique is to calculate a single associated area under the curve of the standard normal distribution and use it to calculate the p-value. For example, in **Figure 2**, if z (the normal equivalent of t) has value -2.0, you can calculate the area from -infinity to -2.0, which is 0.02275. This area under the normal curve can then be used to calculate the corresponding area under the t-distribution.

To summarize, to perform a t-test you must calculate and then sum two (equal) areas under a t-distribution. This area is called the p-value. To do this, you can compute a single area under the standard normal distribution and then use that area to get the p-value.

Calculating the Area Under the Standard Normal Distribution

There are many ways to calculate the area under the standard normal distribution curve. This is one of the oldest problems in computer science. My preferred method is to use what's called ACM algorithm #209. The Association for Computing Machinery (ACM) has published many fundamental algorithms for numerical and statistical computing.

A C# implementation of algorithm #209 is presented in **Figure 4** as function Gauss. The function accepts a value, z, between

-infinity and +infinity and returns a close approximation to the area under the standard normal distribution from -infinity to z.

Even a quick glance at the code in **Figure 4** should convince you that using an existing algorithm, such as ACM #209, is much easier than coding your own implementation from scratch. An alternative to ACM #209 is to use a slight modification of equation 7.1.26 from "Handbook of Mathematical Functions" by Milton Abramowitz and Irene A. Stegun (Dover Publications, 1965).

Calculating the Area Under the T-Distribution

With an implementation of the Gauss function in hand, the area under the t-distribution can be calculated using ACM algorithm #395. A C# implementation of algorithm #395 is presented in **Figure 5** as function Student. The function accepts a t value and a df value and returns the combined area from -infinity to t plus t to +infinity.

Algorithm #395 has two forms. One form accepts the df parameter as an integer value and the second form accepts df as a type double value. In most statistics problems, the degrees of freedom is an integer value, but the Welch t-test uses a type double value.

The Demo Program

To create the demo program, I launched Visual Studio and created a new C# console application named TTest. The demo has no significant .NET version dependencies, so any version of Visual Studio should work. After the template code loaded into the editor, I deleted all using statements except for the single reference to the top-level System namespace. In the Solution Explorer window I renamed file Program.cs to TTestProgram.cs and allowed Visual Studio to automatically rename class Program for me.

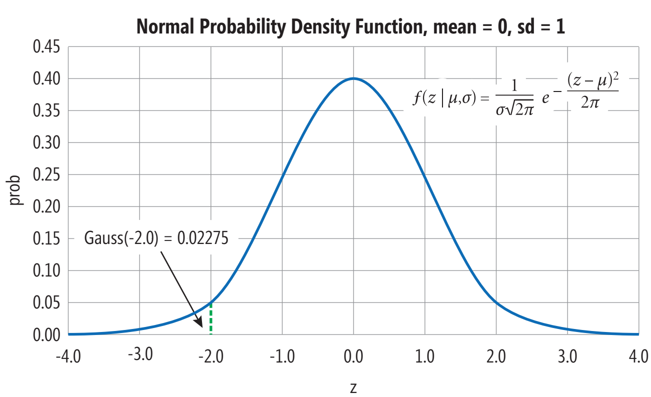


Figure 2 **The Standard Normal Distribution**

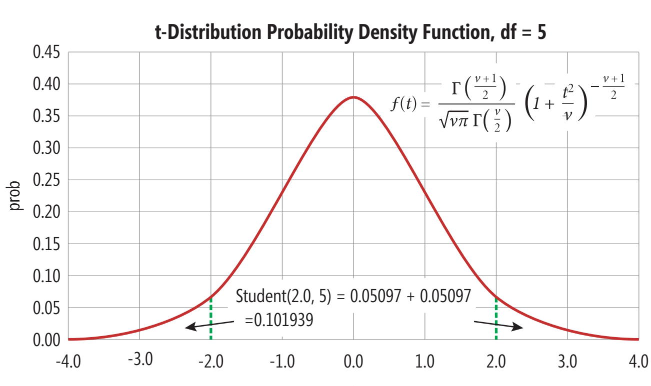


Figure 3 **The T-Distribution**

The demo program is a bit too long to present in its entirety here, but you can find the complete source code in the file download that accompanies this article. The Main method begins by setting up and displaying the two sample datasets:

```
Console.WriteLine("\nBegin Welch's t-test using C# demo\n");
var x = new double[] { 88, 77, 78, 85, 90, 82, 88, 98, 90 };
var y = new double[] { 81, 72, 67, 81, 71, 70, 82, 81 };
Console.WriteLine("\nThe first data set (x) is:\n");
ShowVector(x, 0);
Console.WriteLine("\nThe second data set (y) is:\n");
ShowVector(y, 0);
```

In most statistics problems, the degrees of freedom is an integer value, but the Welch t-test uses a type double value.

All of the work is performed in a method named TTest:

```
Console.WriteLine("\nStarting Welch's t-test using C#\n");
TTest(x, y);
Console.WriteLine("\nEnd t-test demo\n");
Console.ReadLine();
```

The definition of method TTest begins by summing the values in each dataset:

```
public static void TTest(double[] x, double[] y)
{
  double sumX = 0.0;
  double sumY = 0.0;
  for (int i = 0; i < x.Length; ++i)
    sumX += x[i];
  for (int i = 0; i < y.Length; ++i)
    sumY += y[i];
...
```

Next, the sums are used to calculate the two sample means:

```
int n1 = x.Length;
int n2 = y.Length;
double meanX = sumX / n1;
double meanY = sumY / n2;
```

Next, the two means are used to calculate the two sample variances:

```
double sumXminusMeanSquared = 0.0; // Calculate variances
double sumYminusMeanSquared = 0.0;

for (int i = 0; i < n1; ++i)
  sumXminusMeanSquared += (x[i] - meanX) * (x[i] - meanX);

for (int i = 0; i < n2; ++i)
  sumYminusMeanSquared += (y[i] - meanY) * (y[i] - meanY);

double varX = sumXminusMeanSquared / (n1 - 1);
double varY = sumYminusMeanSquared / (n2 - 1);
```

The variance of a set of data is the square of the standard deviation, so the standard deviation is the square root of the variance and the t-test works with variances. Next, the t statistic is calculated:

```
double top = (meanX - meanY);
double bot = Math.Sqrt((varX / n1) + (varY / n2));
double t = top / bot;
```

In words, the t statistic is the difference between the two sample means, divided by the square root of the sum of the variances divided by their associated sample sizes. Next, the degrees of freedom is calculated:

```
double num = ((varX / n1) + (varY / n2)) *
  ((varX / n1) + (varY / n2));
double denomLeft = ((varX / n1) * (varX / n1)) / (n1 - 1);
double denomRight = ((varY / n2) * (varY / n2)) / (n2 - 1);
double denom = denomLeft + denomRight;
double df = num / denom;
```

The calculation of the degrees of freedom for the Welch t-test is somewhat tricky and the equation isn't at all obvious. Fortunately, you'll never have to modify this calculation. Method TTest concludes by computing the p-value and displaying all the calculated values:

```
  ...
    double p = Student(t, df); // Cumulative two-tail density
    Console.WriteLine("mean of x = " + meanX.ToString("F2"));
    Console.WriteLine("mean of y = " + meanY.ToString("F2"));
    Console.WriteLine("t = " + t.ToString("F4"));
    Console.WriteLine("df = " + df.ToString("F3"));
    Console.WriteLine("p-value = " + p.ToString("F5"));
    Explain();
  }
```

The program-defined method named Explain displays information explaining the interpretation of the p-value, as shown in **Figure 1**.

A Few Comments

There are actually several different kinds of statistics problems that involve the t-test. The type of problem described in this article is sometimes called an unpaired t-test because there's no conceptual connection between the data values in each sample dataset. Another type of t-test is called a paired sample test, which might be used when you have some sort of before and after data, such as a test score before some instruction followed by a test score after the instruction. Here, each pair of scores is conceptually related.

Figure 4 **Calculating the Area under the Standard Normal Distribution**

```
public static double Gauss(double z)
{
  // input = z-value (-inf to +inf)
  // output = p under Standard Normal curve from -inf to z
  // e.g., if z = 0.0, function returns 0.5000
  // 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;
}
```

Figure 5 **Calculating the Area under the t-Distribution**

```
public static double Student(double t, double df)
{
  // for large integer df or double df
  // adapted from ACM algorithm 395
  // returns 2-tail p-value
  double n = df; // to sync with ACM parameter name
  double a, b, y;

  t = t * t;
  y = t / n;
  b = y + 1.0;
  if (y > 1.0E-6) y = Math.Log(b);
  a = n - 0.5;
  b = 48.0 * a * a;
  y = a * y;

  y = (((((-0.4 * y - 3.3) * y - 24.0) * y - 85.5) /
    (0.8 * y * y + 100.0 + b) + y + 3.0) / b + 1.0) *
    Math.Sqrt(y);
  return 2.0 * Gauss(-y); // ACM algorithm 209
}
```

The Welch t-test presented here is superior to the more common Student t-test in most scenarios. The Student t-test generally requires an equal number of data points in each of the two sample datasets, and requires that the variances of the two samples be approximately equal. The Welch t-test can work with unequal sample sizes and is robust even when sample variances differ.

The type of t-test explained in this article is called a two-tailed test. This is more or less synonymous with a problem where the goal is to determine whether two group means are the same. A one-tailed t-test can be used in situations where the goal is to determine if the mean of one group is greater than the mean of the second group. When performing a one-tailed t-test, you divide the two-tailed p-value by 2.

You should be very conservative when interpreting the results of a t-test. A conclusion along the lines of, "Based on a calculated t-test p-value of 0.008 I conclude it is unlikely that the true population means of males and females are the same" is much better than, "The p-value of 0.008 means the average scores of males are greater than those of females."

An alternative to the t-test is called the Mann-Whitney U test. Both techniques infer whether two population means are equal or not based on samples, but the Mann-Whitney U test makes fewer statistical assumptions, which leads to more conservative conclusions (you're less likely to conclude the means under investigation are different).

The t-test is limited to situations where there are two groups. For problems examining the means of three of more groups, you'd use an analysis called the F-test. ■

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

Thanks to the following technical expert at Microsoft Research for reviewing this article: *Kirk Olynyk*

TED NEWARD

How To Be MEAN: Express Routing

Welcome back, "Nodeists." (I have no idea if that's the official term for those who use Node.js on a regular basis, but Nodeists sounds better to me than "Nodeheads" or "Noderati" or "Nodeferatu.")

In the last installment (msdn.com/magazine/mt573719), the application's stack had gone from being an "N" stack (just Node) to an "EN" stack by virtue of installing Express to go along with the Node.js. As tempting as it would be to jump directly onto other things, there are a few more things about Express—and its supporting packages and libraries—that deserve exploration and further discussion. You previously got a taste of one of these, Express routing, when the code set up a function to display "Hello World" in response to HTTP requests to the "/" relative URL path. Now, I'll go a little deeper into the Express world and show you how to use it more effectively.

Often, when Nodeists write Express-based applications, they do so in the same manner that we ".NETers" write ASP.NET applications.

By the way, those who are interested in seeing the latest-and-greatest code being written as part of this series can visit the Microsoft Azure site that holds the most recent of this series' code (msdn-mean.azurewebsites.net). It's likely that the information in this column is out of sync with what's on the site, given publication schedules, and the site gives readers a look at what's to come.

Routing, Redux

A recap of the app.js file from the last installment shows the single-endpoint nature of the application that's been built so far, as shown in **Figure 1**, the simple-yet-necessary homage to the Gods of Computer Science.

The part in question is the section of code labeled "Set up a simple route"; here, you're establishing a single endpoint, mapped by the HTTP verb ("get") and relative URL endpoint ("/," passed as the first argument to the "get" method).

It's fairly easy to infer the pattern for the other HTTP verbs—for a "POST" request, you use the post method; for a "PUT," put; and "DELETE," you use delete. Express supports the other verbs, too, but for fairly obvious reasons, these are the four you most care about. Each also then takes as its second argument a function, which in the example in **Figure 1** is a function literal that handles the incoming HTTP request.

Figure 1 **Code for the Express "Hello World"**

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

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

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

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

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

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

});
```

The "E" in MEAN

Often, when Nodeists write Express-based applications, they do so in the same manner that we ".NETers" write ASP.NET applications. The server generates an HTML document containing the presentation (HTML) intermixed with the data, and sends that back to the browser, after which the user fills out a form and POSTs the entered data back to Express; or, the user clicks on a link and generates a GET back at Express to do the complete server-side cycle again. And, because handwriting HTML in Node.js is just as much fun as it is in Visual Basic or C#, a number of tools emerged out of the Node.js world designed to serve the same purpose the Razor syntax does in a classic ASP.NET application. It makes it easier to write the presentation layer without co-mingling the data and code too much.

However, in a MEAN-based application, AngularJS will form the complete client-side experience, so Express takes on the same role as ASP.NET MVC—it's simply a transport layer, taking raw data (usually in the form of JSON) from the client, acting on that data (usually either storing it, modifying it, or finding associated or related data) and sending raw data (again, usually in the form of JSON) back to the client tier. Toward that end, our sojourn in Express will avoid the subject of templating frameworks (of which there are several in the

Node.js world, "handlebars" and "jade" being two of the more popular), and I'll focus explicitly on simply shipping JSON back and forth. Some will call this a RESTful endpoint, but, frankly, REST involves a lot more than just HTTP and JSON, and building a Fielding-approved RESTful system is well beyond the scope of this series.

So, for now, I'll talk about standing up a couple of simple read-only endpoints for any JSON-consuming client to use.

Hello, in JSON

Usually, a Web API follows a fairly loose structure for obtaining data:

- A GET request to a given resource type (such as "persons") will yield a JSON result that's an array of objects, each one containing at minimum a unique identifier (for individual retrieval) and usually some kind of short descriptive text, suitable for display in a list of options.
- A GET request to a given resource type with an identifier as part of the URL ("persons/1234," where 1234 is the identifier uniquely identifying the person we're interested in) will yield a JSON result that is (usually) a single JSON object describing the resource in some level of detail.

Web APIs will also use PUT, POST and DELETE, but for now, I'll focus on just retrieving data.

So, assuming the resource type is "persons," you'll create two endpoints, one labeled "/persons," and the other "/persons/<unique identifier>." For starters, you need a small "database" of persons to work with—first names, last names, and their current "status" (whatever they happen to be doing right now) will suffice (see **Figure 2**).

Not exactly SQL Server, but it'll do for now.

Next, you need the endpoint for the full collection of persons:

```
var getAllPersons = function(req, res) {
  var response = personData;

  res.send(JSON.stringify(response));
};
app.get('/persons', getAllPersons);
```

Notice that in this case, the route mapping is using a standalone function (getAllPersons), which is more common, because it helps keep a separation of concerns a little more clean—the function acts as a controller (in the Model-View-Controller sense). For now, I use JSON.stringify to serialize the array of JavaScript objects into a JSON representation, but I'll use something more elegant later.

Next, you need an endpoint for individual person objects, but this will take a bit more doing because you need to pick up the person identifier as a parameter, and Express has a particular way of doing this. One way (arguably the easier way on the surface of things) is to use the "params" object of the request object (the "req" parameter to the function used in the route map) to fetch the parameter specified in the route, but Node.js can also use a parameter function to do more—it's a form of filter, which will be invoked when a parameter of a particular naming pattern is found:

```
app.get('/persons/:personId', getPerson);

app.param('personId', function (req, res, next, personId) {
  debug("personId found:",personId);
  var person = _.find(personData, function(it) {
    return personId == it.id;
  });
  debug("person:", person);
  req.person = person;
  next();
});
```

However, in a MEAN-based application, AngularJS will form the complete client-side experience, so Express takes on the same role as ASP.NET MVC.

When the route is invoked, whatever follows "/persons" (as in "/persons/1") will be bound into a parameter of name "personId," just as you might find with ASP.NET MVC. But then when using the param function—which will be invoked when any route with ":personId" is invoked—the associated function is invoked, which will look up (using the "lodash" package function find, as shown in the previous code snippet) from the tiny personData database. Then, however, it's added to the "req" object (because JavaScript objects are always dynamically typed, it's trivial to do), so that it will be available to the remainder of what is invoked, which in this case will be the getPerson function—this now becomes quite trivial, because the object you want to return is already fetched:

```
var getPerson = function(req, res) {
  if (req.person) {
    res.send(200, JSON.stringify(req.person));
  }
  else {
    res.send(400, { message: "Unrecognized identifier: " + identifier });
  }
};
```

See what I mean by "trivial"?

Wrapping Up

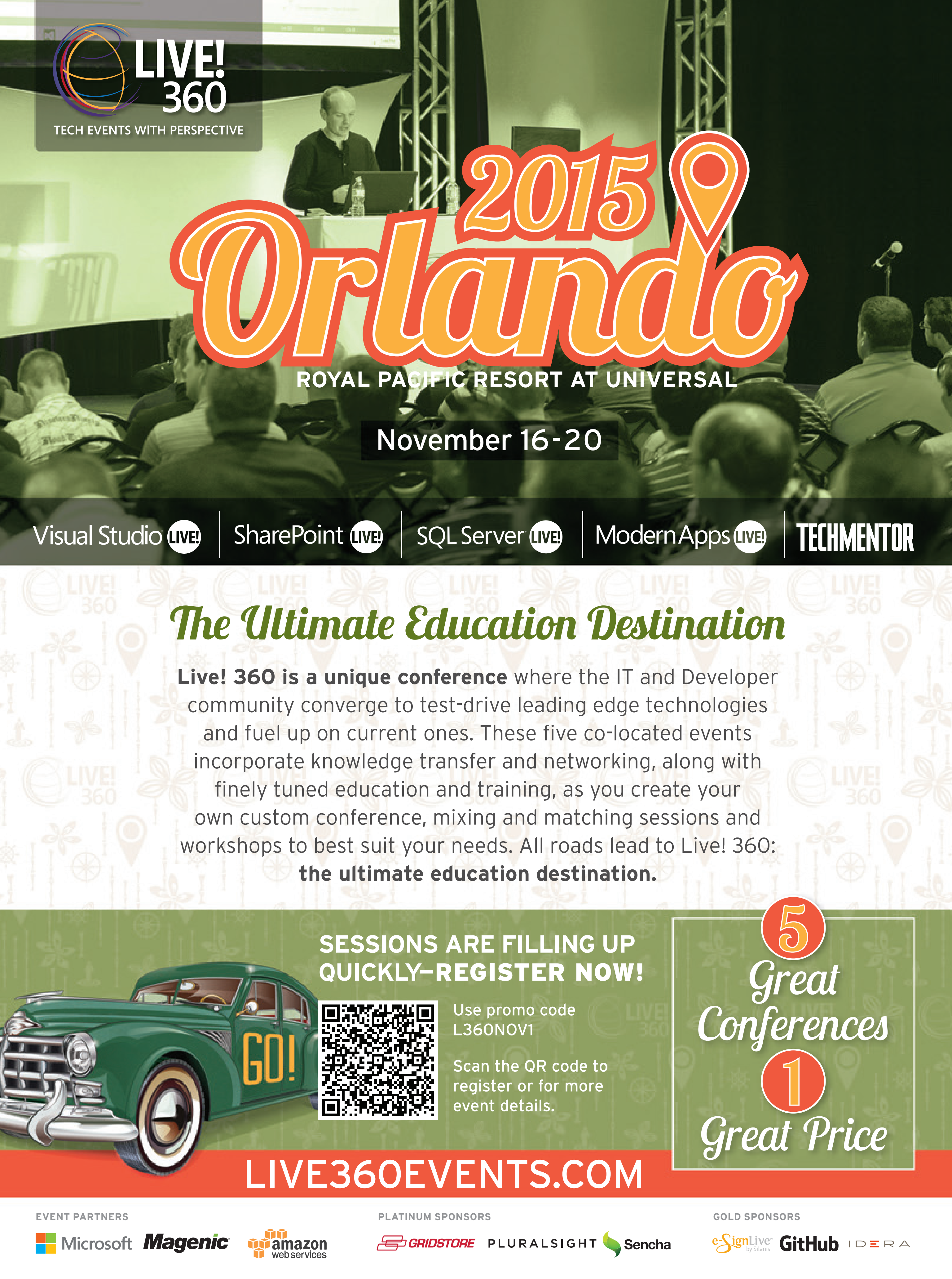
I've got a bit more to do with Express, but despite being on a roll here, I'm out of space for this one, so … happy coding! ■

Ted Neward *is the CTO at iTrellis, a consulting services company. He has written more than 100 articles and authored or co-authored a dozen books, including "Professional F# 2.0" (Wrox, 2010). He is an F# MVP and speaks at conferences around the world. He consults and mentors regularly—reach him at ted@tedneward.com or ted@itrellis.com if you're interested.*

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

Figure 2 **Creating a Small Database of Persons**

```
var personData = [
  {
    "id": 1,
    "firstName": "Ted",
    "lastName": "Neward",
    "status": "MEANing"
  },
  {
    "id": 2,
    "firstName": "Brian",
    "lastName": "Randell",
    "status": "TFSing"
  }
];
```


LIVE! 360
TECH EVENTS WITH PERSPECTIVE
2015
Orlando
ROYAL PACIFIC RESORT AT UNIVERSAL
November 16-20
Visual Studio LIVE!
SharePoint LIVE!
SQL Server LIVE!
Modern Apps LIVE!
TECHMENTOR
The Ultimate Education Destination
Live! 360 is a unique conference where the IT and Developer community converge to test-drive leading edge technologies and fuel up on current ones. These five co-located events incorporate knowledge transfer and networking, along with finely tuned education and training, as you create your own custom conference, mixing and matching sessions and workshops to best suit your needs. All roads lead to Live! 360: **the ultimate education destination.**
SESSIONS ARE FILLING UP QUICKLY—REGISTER NOW!
GO!
Use promo code L360NOV1
Scan the QR code to register or for more event details.
5 Great Conferences
1 Great Price
LIVE360EVENTS.COM
EVENT PARTNERS
Microsoft
Magenic
amazon web services
PLATINUM SPONSORS
GRIDSTORE
PLURALSIGHT
Sencha
GOLD SPONSORS
eSignLive by Silanis
GitHub
IDERA

MARK MICHAELIS

C# Exception Handling

Welcome to the inaugural Essential .NET column. It's here where you'll be able to follow all that is happening in the Microsoft .NET Framework world, whether it's advances in C# vNext (currently C# 7.0), improved .NET internals, or happenings on the Roslyn and .NET Core front (such as MSBuild moving to open source).

I've been writing and developing with .NET since it was announced in preview in 2000. Much of what I'll write about won't just be the new stuff, but about how to leverage the technology with an eye toward best practices.

I live in Spokane, Wash., where I'm the "Chief Nerd" for a high-end consulting company called IntelliTect (IntelliTect.com). IntelliTect specializes in developing the "hard stuff," with excellence. I've been a Microsoft MVP (currently for C#) going on 20 years and a Microsoft regional director for eight of those years. Today, this column launches with a look at updated exception handling guidelines.

C# 6.0 included two new exception handling features. First, it included support for exception conditions—the ability to provide an expression that filters out an exception from entering catch block before the stack unwinds. Second, it included async support from within a catch block, something that wasn't possible in C# 5.0 when async was added to the language. In addition, there have been many other changes that have occurred in the last five versions of C# and the corresponding .NET Framework, changes, which in some cases, are significant enough to require edits to C# coding guidelines. In this installment, I'll review a number of these changes and provide updated coding guidelines as they relate to exception handling—catching exceptions.

Catching Exceptions: Review

As is fairly well understood, throwing a particular exception type enables the catcher to use the exception's type itself to identify the problem. It's not necessary, in other words, to catch the exception and use a switch statement on the exception message to determine which action to take in light of the exception. Instead, C# allows for multiple catch blocks, each targeting a specific exception type as shown in **Figure 1**.

When an exception occurs, the execution will jump to the first catch block that can handle it. If there is more than one catch block associated with the try, the closeness of a match is determined by the inheritance chain (assuming no C# 6.0 exception condition) and the first one to match will process the exception. For example, even though the exception thrown is of type System.Exception, this "is a" relationship occurs through inheritance because System.InvalidOperationException ultimately derives from System.Exception.

Figure 1 **Catching Different Exception Types**

```
using System;

public sealed class Program
{
  public static void Main(string[] args)
    try
    {
      / ...
      throw new InvalidOperationException(
          "Arbitrary exception");
      // ...
    }
    catch(System.Web.HttpException exception)
      when(exception.GetHttpCode() == 400)
    {
      // Handle System.Web.HttpException where
      // exception.GetHttpCode() is 400.
    }
    catch (InvalidOperationException exception)
    {
      bool exceptionHandled=false;
      // Handle InvalidOperationException
      // ...
      if(!exceptionHandled)
        // In C# 6.0, replace this with an exception condition
      {
        throw;
      }
    }
    finally
    {
      // Handle any cleanup code here as it runs
      // regardless of whether there is an exception
    }
  }
}
```

Because InvalidOperationException most closely matches the exception thrown, catch(InvalidOperationException...) will catch the exception and not catch(Exception...) block if there was one.

Catch blocks must appear in order (again assuming no C# 6.0 exception condition), from most specific to most general, to avoid a compile-time error. For example, adding a catch(Exception...) block before any of the other exceptions will result in a compile error because all prior exceptions derive from System.Exception at some point in their inheritance chain. Also note that a named parameter for the catch block is not required. In fact, a final catch without even the parameter type is allowed, unfortunately, as discussed under general catch block.

On occasion, after catching an exception, you might determine that, in fact, it isn't possible to adequately handle the exception. Under this scenario, you have two main options. The first option is to rethrow a different exception. There are three common scenarios for when this might make sense:


WASHINGTON, DC
JUNE 8-9 2016
ACQUIRE
Acquisition & Management Show
ACQUIRE is a new 2-day event that focuses on 3 key OMB spending categories—Professional Services, Office Management and Information Technology. Covering all aspects of the acquisition and management process, from setting policy and defining requirements to implementing and managing programs to end user experience, it's guaranteed to be THE NEXT BIG THING.
Start planning your presence now.
Exhibit & Sponsorship packages available.
Contact Stacy Money for pricing & details:
smoney@1105media.com | 415.444.6933
ACQUIREshow.com
Tracks include:
ACQUIRE Office Management
ACQUIRE Professional Services
ACQUIRE Information Technology
FEDERAL IT ACQUISITION SUMMIT
ACQUIRE Acquisition Management
ACQUIRE Project Management
ACQUIRE Industry Days
Washington Technology
ACQUIRE Happy Fed
Federal SOUP

Scenario No. 1 The captured exception does not sufficiently identify the issue that triggered it. For example, when calling System.Net.WebClient.DownloadString with a valid URL, the runtime might throw a System.Net.WebException when there's no network connection—the same exception thrown with a non-existent URL.

Scenario No. 2 The captured exception includes private data that should not be exposed higher up the call chain. For example, a very early version of CLR v1 (pre-alpha, even) had an exception that said something like, "Security exception: You do not have permission to determine the path of c:\temp\foo.txt."

Scenario No. 3 The exception type is too specific for the caller to handle. For example, a System.IO exception (such as UnauthorizedAccessException IOException FileNotFoundException DirectoryNotFoundException PathTooLongException, NotSupportedException or SecurityException ArgumentException) occurs on the server while invoking a Web service to look up a ZIP code.

When rethrowing a different exception, pay attention to the fact that it could lose the original exception (presumably intentionally in the case of Scenario 2). To prevent this, set the wrapping exception's InnerException property, generally assignable via the constructor, with the caught exception unless doing so exposes private data that shouldn't be exposed higher in the call chain. In so doing, the original stack trace is still available.

If you don't set the inner exception and yet still specify the exception instance after the throw statement (throw exception) the location stack trace will be set on the exception instance. Even if you rethrow the exception previously caught, whose stack trace is already set, it will be reset.

A second option when catching an exception is to determine that, in fact, you cannot appropriately handle it. Under this scenario you will want to rethrow the exact same exception—sending it to the next handler up the call chain. The InvalidOperationException catch block of **Figure 1** demonstrates this. A throw statement appears without any identification of the exception to throw (throw is on its own), even though an exception instance (exception) appears in the catch block scope that could be rethrown. Throwing a specific exception would update all the stack information to match the new throw location. As a result, all the stack information indicating the call site where the exception originally occurred would be lost, making it significantly more difficult to diagnose the problem. Upon determining that a catch block cannot sufficiently handle an exception, the exception should be rethrown using an empty throw statement.

Figure 2 **Using ExceptionDispatchInfo to Rethrow an Exception**

```
using System
using System.Runtime.ExceptionServices;
using System.Threading.Tasks;
Task task = WriteWebRequestSizeAsync(url);
try
{
  while (!task.Wait(100))
{
    Console.Write(".");
  }
}
catch(AggregateException exception)
{
  exception = exception.Flatten();
  ExceptionDispatchInfo.Capture(
    exception.InnerException).Throw();
}
```

Regardless of whether you're rethrowing the same exception or wrapping an exception, the general guideline is to *avoid exception reporting or logging lower in the call stack*. In other words, don't log an exception every time you catch and rethrow it. Doing so causes unnecessary clutter in the log files without adding value because the same thing will be recorded each time. Furthermore, the exception includes the stack trace data of when it was thrown, so no need to record that each time. By all means log the exception whenever it's handled or, in the case that it's not going to be handled, log it to record the exception before shutting down a process.

Throwing Existing Exceptions Without Replacing Stack Information

In C# 5.0, a mechanism was added that enables the throwing of a previously thrown exception without losing the stack trace information in the original exception. This lets you rethrow exceptions, for example, even from outside a catch block and, therefore, without using an empty throw. Although it's fairly rare to need to do this, on some occasions exceptions are wrapped or saved until the program execution moves outside the catch block. For example, multithreaded code might wrap an exception with an AggregateException. The .NET Framework 4.5 provides a System.Runtime.ExceptionServices.ExceptionDispatchInfo class specifically to handle this scenario through the use of its static Capture and instance Throw methods. **Figure 2** demonstrates rethrowing the exception without resetting the stack trace information or using an empty throw statement.

With the ExeptionDispatchInfo.Throw method, the compiler doesn't treat it as a return statement in the same way that it might a normal throw statement. For example, if the method signature returned a value but no value was returned from the code path with ExceptionDispatchInfo.Throw, the compiler would issue an error indicating no value was returned. On occasion, developers might be forced to follow ExceptionDispatchInfo.Throw with a return statement even though such a statement would never execute at run time—the exception would be thrown instead.

Catching Exceptions in C# 6.0

The general exception handling guideline is to *avoid catching exceptions that you're unable to address fully*. However, because catch expressions prior to C# 6.0 could only filter by exception type, the ability to check the exception data and context prior to unwinding the stack at the catch block required the catch block to become the handler for the exception before examining it. Unfortunately, upon determining not to handle the exception, it's cumbersome to write code that allows a different catch block within the same context to handle the exception. And, rethrowing the same exception results in having to invoke the two-pass exception process again, a process that involves first delivering the exception up the call chain until it finds one that handles it and, second, unwinding the call stack for each frame between the exception and the catch location.

Once an exception is thrown, rather than unwinding the call stack at the catch block only to have the exception rethrown because further examination of the exception revealed it couldn't be sufficiently handled, it would obviously be preferable not to catch the exception in the first place. Starting with C# 6.0, an additional

msdn magazine VP

MSDN MAGAZINE VENDOR PROFILE

1&1®

1&1 Cloud Server

Top-performing and dynamic, offering all the advantages of dedicated hardware performance combined with the flexibility of the cloud.

Powered by intel® Cloud Technology

Top performance tested by CLOUD SPECTATOR

Top Performer as Tested by Cloud Spectator

Cloud Spectator, a cloud performance analysis agency, evaluated the 1&1 Cloud Server against other major competitors and based off of raw-performance and price-performance benchmarks found that our product demonstrated strong outputs in both categories, often superior to that of the other tested servers.

Top-Notch Security and Transparent Costs

All servers come with a built-in firewall to help protect against all online threats, the ability to implement backups and take snapshots to prevent accidental data loss, and are housed in data centers that are amongst the safest in the US.

Thanks to a clearly structured cost overview, you'll understand where your money is going and how it's being utilized, enabling you to more efficiently plan and manage your projects. No minimum contract term is required.

Fully Customizable Configuration of SSD, RAM and CPU

You can independently adjust your SSD, RAM, and CPU according to the needs of your business, and you also have the option to switch, shut down, or adjust your package or adjust your package on a by-the-minute basis to meet your changing needs. We also have discounted pre-configured packages available.

Best Performance and Apps

1&1 Cloud Panel gives you an innovative, user-friendly interface with smart administration. You'll be able to make adjustments to your settings at any time. The 1&1 Cloud Server also includes Plesk 12, which is the world's leading software for professional server administration.

Popular open source applications like WordPress, Drupal, Magento and more can be downloaded and installed in just a few clicks.

To learn more please visit our website → www.1and1.com

or send an e-mail to receive a special offer: msdn-voucher@1and1.com

conditional expression is available for catch blocks. Rather than limiting whether a catch block matches based only on an exception type match, C# 6.0 includes support for a conditional clause. The when clause lets you supply a Boolean expression that further filters the catch block to only handle the exception if the condition is true. The System.Web.HttpException block in **Figure 1** demonstrated this with an equality comparison operator.

An interesting outcome of the exception condition is that, when an exception condition is provided, the compiler doesn't force catch blocks to appear in the order of the inheritance chain. For example, a catch of type System.ArgumentException with an accompanying exception condition can now appear before the more specific System.ArgumentNullException type even though the latter derives from the former. This is important because it lets you write a specific exception condition that's paired to a general exception type followed by a more specific exception type (with or without an exception condition). The runtime behavior remains consistent with earlier versions of C#; exceptions are caught by the first catch block that matches. The added complexity is simply that whether a catch block matches is determined by the combination of the type and the exception condition and the compiler only enforces order relative to catch blocks without exception conditions. For example, a catch(System.Exception) with an exception condition can appear before a catch(System.ArgumentException) with or without an exception condition. However, once a catch for an exception type without an exception condition appears, no catch of a more specific exception block (say catch(System.ArgumentNullException)) may occur whether it has an exception condition. This leaves the programmer with the "flexibility" to code exception conditions that are potentially out of order—with earlier exception conditions catching exceptions intended for the later ones, potentially even rendering the later ones unintentionally unreachable. Ultimately, the order of your catch blocks is similar to the way you would order if-else statements. Once the condition is met, all other catch blocks are ignored. Unlike the conditions in an if-else statement, however, all catch blocks must include the exception type check.

Updated Exception Handling Guidelines

The comparison operator example in **Figure 1** is trivial, but the exception condition isn't limited to simplicity. You could, for example, make a method call to validate a condition. The only requirement is that the expression is a predicate—it returns a Boolean value. In other words, you can essentially execute any code you like from within the catch exception call chain. This opens up the possibility of never again catching and rethrowing the same exception again; essentially, you're able to narrow down the context sufficiently before catching the exception as to only catch it when doing so is valid. Thus, the guideline to avoid catching exceptions that you're unable to handle fully becomes a reality. In fact, any conditional check surrounding an empty throw statement can likely be flagged with a code smell and avoided. *Consider adding an exception condition in favor of having to use an empty throw statement except to persist a volatile state before a process terminates*.

That said, developers should limit conditional clauses to check the context only. This is important because if the conditional expression itself throws an exception, then that new exception is ignored and the condition is treated as false. For this reason, you should *avoid throwing exceptions in the exception conditional expression*.

General Catch Block

C# requires that any object that code throws must derive from System.Exception. However, this requirement isn't universal to all languages. C/C++, for example, lets any object type be thrown, including managed exceptions that don't derive from System.Exception or even primitive types like int or string. Starting with C# 2.0, all exceptions, whether deriving from System.Exception or not, will propagate into C# assemblies as derived from System.Exception. The result is that System.Exception catch blocks will catch all "reasonably handled" exceptions not caught by earlier blocks. Prior to C# 1.0 however, if a non-System.Exception-derived exception was thrown from a method call (residing in an assembly not written in C#), the exception wouldn't be caught by a catch(System.Exception) block. For this reason, C# also supports a general catch block (catch{}) that now behaves identically to the catch(System.Exception exception) block except that there's no type or variable name. The disadvantage of such a block is simply that there's no exception instance to access, and therefore no way to know the appropriate course of action. It wouldn't even be possible to log the exception or recognize the unlikely case where such an exception is innocuous.

In practice, the catch(System.Exception) block and general catch block—herein generically referred to as catch System.Exception block—are both to be avoided except under the pretense of "handling" the exception by logging it before shutting down the process. Following the general principle of only catch exceptions that you can handle, it would seem presumptuous to write code for which the programmer declares—this catch can handle any and all exceptions that may be thrown. First, the effort to catalog any and all exceptions (especially in the body of a Main where the amount of executing code is the greatest and context likely the least) seems monumental except for the simplest of programs. Second, there's a host of possible exceptions that can be unexpectedly thrown.

Prior to C# 4.0 there was a third set of corrupted state exceptions for which a program was not even generally recoverable. This set is less of a concern starting in C# 4.0, however, because catching System.Exception (or a general catch block) will not in fact catch such exceptions. (Technically you can decorate a method with the System.Runtime.ExceptionServices.HandleProcessCorruptedStateExceptions so that even these exceptions are caught, but the likelihood that you can sufficiently address such exceptions is extremely challenging. See `bit.ly/1FgeCU6` for more information.)

One technicality to note on corrupted state exceptions is that they will only pass over catch System.Exception blocks when thrown by the runtime. An explicit throw of a corrupted state exception such as a System.StackOverflowException or other System.SystemException will in fact be caught. However, throwing such would be extremely misleading and is really only supported for backward-compatibility reasons. Today's guideline is not to throw any of the corrupted state exceptions (including System.StackOverflowException, System.SystemException, System.OutOfMemoryException, System.Runtime.InteropServices.COMException, System.Runtime.InteropServices.SEHException and System.ExecutionEngineException).

In summary, avoid using a catch System.Exception block unless it's to handle the exception with some cleanup code and logging the exception before rethrowing or gracefully shutting down the application. For example, if the catch block could successfully save any volatile data (something that cannot necessarily be assumed as it, too, might be corrupt) before shutting down the application or rethrowing the exception. When encountering a scenario for which the application should terminate because it's unsafe to continue execution, code should invoke the System.Environment.FailFast method. *Avoid System.Exception and general catch blocks except to gracefully log the exception before shutting down the application*.

Wrapping Up

In this article I provided updated guidelines for exception handling—catching exceptions, updates caused by improvements in C# and the .NET Framework that have occurred over the last several versions. In spite of the fact that there were some new guidelines, many are still just as firm as before. Here's a summary of the guidelines for catching exceptions:

- AVOID catching exceptions that you're unable to handle fully.
- AVOID hiding (discarding) exceptions you don't fully handle.
- DO use throw to rethrow an exception; rather than throw <exception object> inside a catch block.
- DO set the wrapping exception's InnerException property with the caught exception unless doing so exposes private data.
- CONSIDER an exception condition in favor of having to rethrow an exception after capturing one you can't handle.
- AVOID throwing exceptions from exception conditional expression.
- DO use caution when rethrowing different exceptions.
- Rarely use System.Exception and general catch blocks—except to log the exception before shutting down the application.
- AVOID exception reporting or logging lower in the call stack.

Go to itl.tc/ExceptionGuidelinesForCSharp for a review of the details of each of these. In a future column I plan to focus more on the guidelines for throwing exceptions. Suffice it to say for now that a theme for throwing exceptions is: The intended recipient of an exception is a programmer rather than the end user of a program.

Note that much of this material is taken from the next edition of my book, "Essential C# 6.0 (5th Edition)" (Addison-Wesley, 2015), which is available now at itl.tc/EssentialCSharp. ■

Mark Michaelis *is founder of IntelliTect, where he serves as its chief technical architect and trainer. For nearly two decades he has been a Microsoft MVP, and a Microsoft Regional Director since 2007. Michaelis serves on several Microsoft software design review teams, including C#, Microsoft Azure, SharePoint and Visual Studio ALM. He speaks at developer conferences and has written numerous books including his most recent, "Essential C# 6.0 (5th Edition)." Contact him on Facebook at facebook.com/Mark.Michaelis, on his blog at IntelliTect.com/Mark, on Twitter: @markmichaelis or via e-mail at mark@IntelliTect.com.*

Thanks to the following technical experts for reviewing this article:
Kevin Bost, Jason Peterson and Mads Torgerson

Alan Turing and Ashley Madison

What on earth could these names have to do with each other? One symbolizes our highest intellect, the other our most primal animal instincts. Are they not polar opposites? As always, Plattski notices similarities where no one else does.

No one on the Internet could miss the news that AshleyMadison.com (a Web site claiming to connect consenting adults for extra-marital affairs) got hacked and its list of subscribers published. Many users stupid enough to use their real identities had to scramble for cover. See, for example, the article "My Pastor Is on the Ashley Madison List," by Ed Stetzer in *Christianity Today* (bit.ly/1NOHjsn). The rest of the world chuckled with *schadenfreude*, as you, dear reader, are doing right now. (No? Liar.)

Hooray! We've passed the Turing test, at least for desperate guys being told what they sorely want to hear.

But the real surprise is that, according to Annalee Newitz at Gizmodo.com, most of the male subscribers were corresponding not with actual female people, but with automated conversation bots. Once a user set up a free account, these bots initiated conversations and sent messages, trying to tempt the guy to upgrade to a paid membership. Writes Newitz (bit.ly/1OULHqw): "Ashley Madison created more than 70,000 female bots to send male users millions of fake messages, hoping to create the illusion of a vast playland of available women." (*Now* you're chuckling, and don't tell me you're not.)

The notion of a conversation bot that can fool human subjects has been around for a while. You've probably seen the phony psychiatrist program Eliza, first developed by Joseph Weizenbaum in 1966, which bluffed its way through the role of a psychotherapist simply by matching patterns in its input and changing verb conjugations. (Patient: "I am unhappy." Eliza: "Did you come to me because you are unhappy?" One wonders exactly what percentage of human-delivered psychotherapy works this same way.)

We geeks all know the Turing test, proposed by Turing himself to detect artificial intelligence. Have a conversation over a terminal, and if you can't tell whether you're talking to a human or a program, then that program is intelligent. In a 2014 contest sponsored by the Royal Society of London, the chatterbot Eugene Goostman managed to fool 33 percent of the judges in a 5-minute conversation.

But the calculus changes when the human subjects *want* to be deceived. As Weizenbaum wrote in his original 1966 paper describing the Eliza experiment, subjects actively participated in their own deception (bit.ly/1G6UAGb). The user's belief that the bot is a real person, Weizenbaum wrote, "serves the speaker to maintain his sense of being heard and understood. The speaker further defends his impression (which even in real life may be illusory) by attributing to his conversational partner [the computer] all sorts of background knowledge, insights and reasoning ability." In short, some subjects *refused to believe Eliza was faking her responses, even after they were told*.

The same applies to Ashley Madison. The customers wanted to find something enticing, so find it they did. Hooray! We've passed the Turing test, at least for desperate guys being told what they sorely want to hear.

The Ashley Madison bots communicated by written messages. But with large-scale voice recognition now in the mainstream (see Siri and Cortana), I foresee the tipping point at which bots replace human adult phone workers. Imagine: Customer (speaking on phone): "I have this thing for waffle irons." Audio chatterbot: "Did you come to me because you have this thing for waffle irons?" You'll be able to choose the bot's output voice, as you can today for your GPS directions—how about phone calls from Taylor Swift, Kathleen Turner or James Earl Jones? French accent? *Mais oui*. One wonders how long until supercomputers provide on-demand real-time video synthesis, as supercomputer Mike Holmes did with his human video appearance in the Robert Heinlein novel, "The Moon Is a Harsh Mistress." The 2013 movie "She," in which a user falls in love with his digital assistant, shows the logical progression of this idea. (I could develop it toward peripheral devices, but my editor won't let me. "Hold it right there," he says. To which I reply, "Precisely.")

Let's face it: If we're looking for intelligence, using one's real identity on an inherently shady site is a counter marker. Maybe we should think of the Ashley Madison leak as a reverse Turing test. ■

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.*

Aspose.Cells

Work with spreadsheets and data without depending on Microsoft Excel

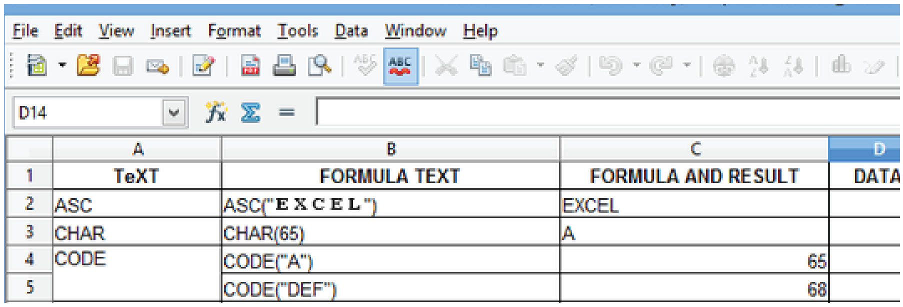
- Solution for spreadsheet creation, manipulation and conversion.
- Import and export data.

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

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

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

A flexible API for simple and complex spreadsheet programming.



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

Common Uses

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

Key Features

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

Supported File Formats

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

Format support varies across platforms.

Platforms

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

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

Aspose.Words

Program with word processing documents independently of Microsoft Word

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

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

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

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

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

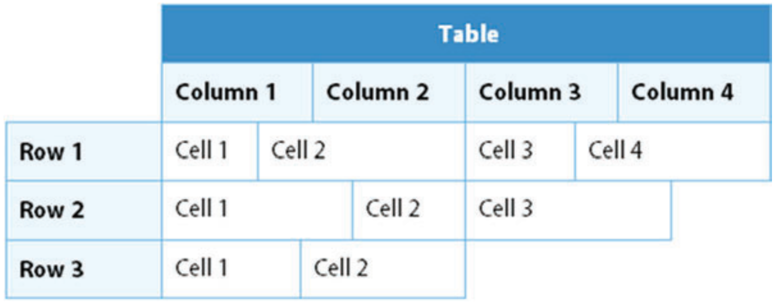


	Table			
	Column 1	Column 2	Column 3	Column 4
Row 1	Cell 1	Cell 2	Cell 3	Cell 4
Row 2	Cell 1	Cell 2	Cell 3	
Row 3	Cell 1	Cell 2		

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

Common Uses

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

Key Features

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

Supported File Formats

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

Format support varies across platforms.

Platforms

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

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

Case Study: Aspose.Words for .NET

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

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

Problem

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

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

Looking for a Solution

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

Implementation

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

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

Outcome

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

Next Steps

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

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

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

Adding File Conversion and Manipulation to Business Systems

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

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

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

Aspose creates APIs that work independently of Microsoft Office Automation.

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

Document Conversion Options

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

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

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

Look to Aspose

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

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

Finding the Right Tool

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

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

EU: +44 141 416 1112

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

Oceania: +61 2 8003 5926

Aspose.BarCode

A complete toolkit for barcode generation and recognition

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

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

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

Robust and reliable barcode generation and recognition.

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

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

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

Common Uses

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

Key Features

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

Supported File Formats

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

Format support varies across platforms.

Supported Barcodes

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

Platforms

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

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

EU: +44 141 416 1112

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

Oceania: +61 2 8003 5926

Aspose.Email

Work with emails and calendars without Microsoft Outlook

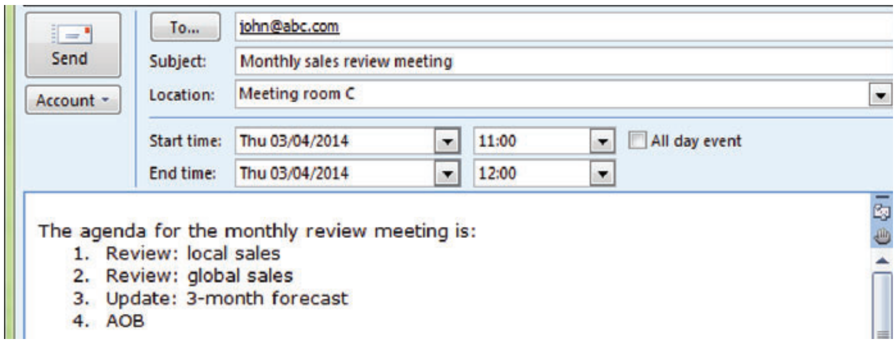
- Complete email processing solution.
- Message file format support.

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

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

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

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



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

Common Uses

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

Key Features

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

Supported File Formats

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

Format support varies across platforms.

Platforms

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

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

www.aspose.com

EU: +44 141 416 1112

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

Oceania: +61 2 8003 5926

Aspose.Pdf

Create PDF documents without using Adobe Acrobat

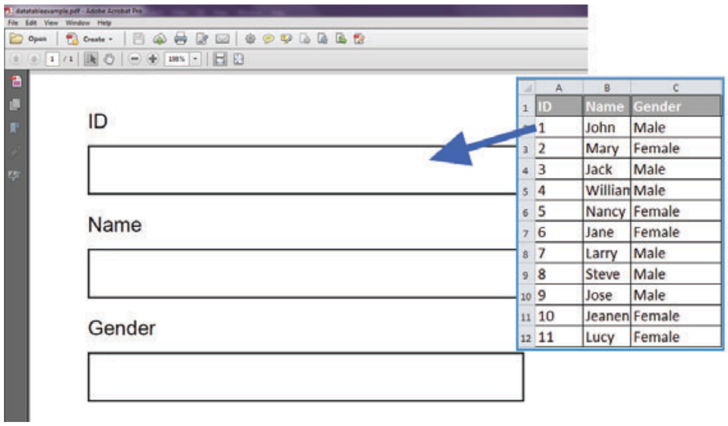
- A complete solution for programming with PDF files.
- Work with PDF forms and form fields.

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

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

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

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



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

Common Uses

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

Key Features

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

Format support varies across platforms.

Platforms

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

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

Conversion is Fast And High-Fidelity

Aspose.Note for .NET

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

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

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

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

Features

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

Aspose.Imaging

Create Images from scratch.

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

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

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

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

GIF BMP JPEG TIFF

PNG PDF DXF DWG PSD

Aspose.Imaging allows creation and manipulation of images.

Common Uses

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

Key Features

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

Supported File Formats

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

Platforms

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

The pricing info above is for .NET.

Aspose.Slides

Work with presentations without using Microsoft PowerPoint

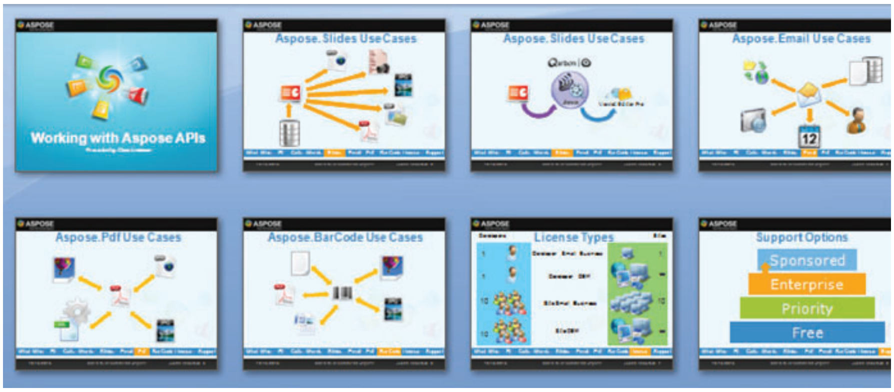
- Complete solution for working with presentation files.
- Export presentations and slides to portable or image formats.

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

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

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

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



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

Common Uses

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

Key Features

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

Supported File Formats

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

Format support varies across platforms.

Platforms

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

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

Support Services

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

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

NO ONE KNOWS OUR PRODUCTS AS WELL AS WE DO. We develop them, support them and use them. Our experience is available to you, whether you want us to develop a solution for you, or you just need a little help to solve a particular problem.

Consulting

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

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

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

Work with the most experienced Aspose developers in the world.

Consulting Benefits

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

Support Options

Free

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

Priority

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

Enterprise

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

Pricing Info
Each consulting project is evaluated individually; no two projects have exactly the same requirements.
To see the Priority and Enterprise support rates, refer to the product price list, or contact our sales team.