THE MICROSOFT JOURNAL FOR DEVELOPERS

APRIL 2014 VOL 29 NO 4

msdn magazine

C++

THE MICROSOFT JOURNAL FOR DEVELOPERS

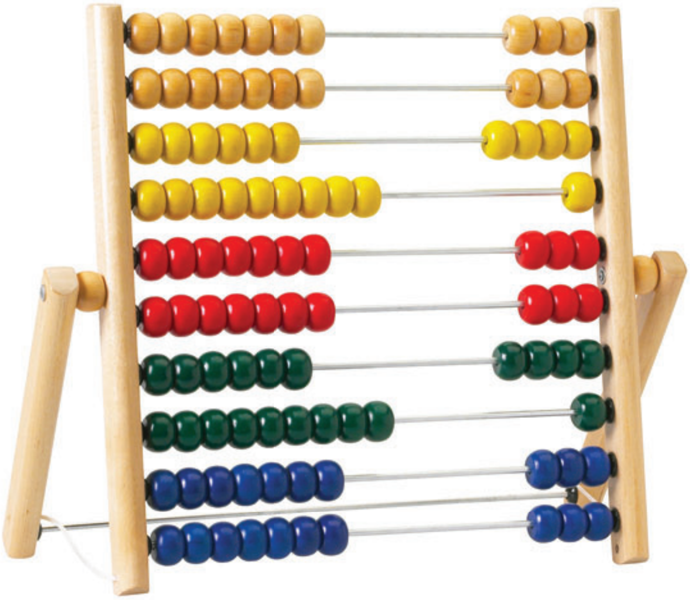
APRIL 2014 VOL 29 NO 4

msdn magazine

C++

COLUMNS

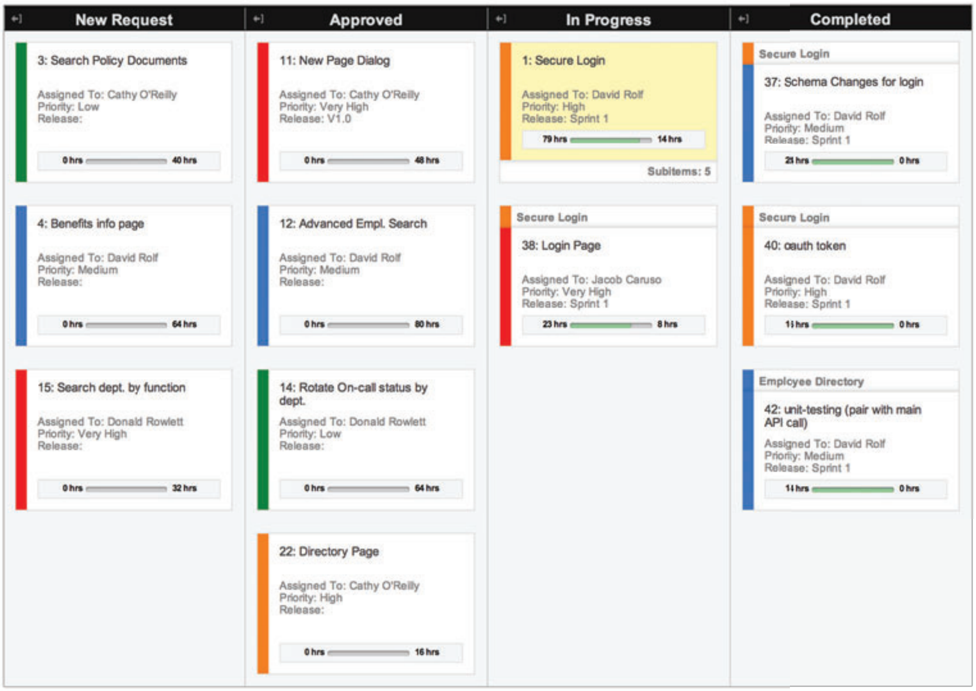
Microsoft



VS.

VS.

OnTime THE #1 SELLING SCRUM SOFTWARE



New Request

3: Search Policy Documents
Assigned To: Cathy O'Reilly
Priority: Low
Release:
0 hrs 40 hrs

4: Benefits info page
Assigned To: David Rolf
Priority: Medium
Release:
0 hrs 64 hrs

15: Search dept. by function
Assigned To: Donald Rowlett
Priority: Very High
Release:
0 hrs 32 hrs

Approved

11: New Page Dialog
Assigned To: Cathy O'Reilly
Priority: Very High
Release: V1.0
0 hrs 48 hrs

12: Advanced Empl. Search
Assigned To: David Rolf
Priority: Medium
Release:
0 hrs 80 hrs

14: Rotate On-call status by dept.
Assigned To: Donald Rowlett
Priority: Low
Release:
0 hrs 64 hrs

22: Directory Page
Assigned To: Cathy O'Reilly
Priority: High
Release:
0 hrs 16 hrs

In Progress

1: Secure Login
Assigned To: David Rolf
Priority: High
Release: Sprint 1
79 hrs 14 hrs
Subitems: 5

Secure Login
38: Login Page
Assigned To: Jacob Caruso
Priority: Very High
Release: Sprint 1
23 hrs 8 hrs

Completed

Secure Login
37: Schema Changes for login
Assigned To: David Rolf
Priority: Medium
Release: Sprint 1
21 hrs 0 hrs

Secure Login
40: oauth token
Assigned To: David Rolf
Priority: High
Release: Sprint 1
15 hrs 0 hrs

Employee Directory
42: unit-testing (pair with main API call)
Assigned To: David Rolf
Priority: Medium
Release: Sprint 1
14 hrs 0 hrs

Boosting your Scrum team's efficiency is easy. Make the switch from sticky notes and upgrade to OnTime at **OnTimeNow.com/MSDN.**

msdn magazine

APRIL 2014 VOLUME 29 NUMBER 4

MOHAMMAD AL-SABT Editorial Director/mmeditor@microsoft.com
KENT SHARKEY Site Manager

MICHAEL DESMOND Editor in Chief/mmeditor@microsoft.com
LAFE LOW Features Editor
SHARON TERDEMAN Features Editor
DAVID RAMEL Technical Editor
WENDY HERNANDEZ Group Managing Editor

SCOTT SHULTZ Creative Director
JOSHUA GOULD Art Director

SENIOR CONTRIBUTING EDITOR Dr. James McCaffrey
CONTRIBUTING EDITORS Rachel Appel, Dino Esposito, Kenny Kerr, Julie Lerman, Ted Neward, Charles Petzold, David S. Platt, Bruno Terkaly, Ricardo Villalobos

RedmondMediaGroup

Henry Allain President, Redmond Media Group
Michele Imgrund Sr. Director of Marketing & Audience Engagement
Tracy Cook Director of Online Marketing
Irene Fincher Audience Development Manager

ADVERTISING SALES: 818-674-3416/dlabianca@1105media.com

Dan LaBianca Vice President, Group Publisher
Chris Kourtoglou Regional Sales Manager
Danna Vedder Regional Sales Manager/Microsoft Account Manager
David Seymour Director, Print & Online Production
Anna Lyn Bayusa Production Coordinator/msdnadproduction@1105media.com

1105 MEDIA INC

Neal Vitale President & Chief Executive Officer
Richard Vitale Senior Vice President & Chief Financial Officer
Michael J. Valenti Executive Vice President

Christopher M. Coates Vice President, Finance & Administration
Erik A. Lindgren Vice President, Information Technology & Application Development
David F. Myers Vice President, Event Operations

Jeffrey S. Klein Chairman of the Board

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.

Printed in the U.S.A. Reproductions in whole or part prohibited except by written permission. Mail requests to "Permissions Editor," c/o *MSDN Magazine*, 4 Venture, Suite 150, Irvine, CA 92618.

Legal Disclaimer: The information in this magazine has not undergone any formal testing by 1105 Media, Inc. and is distributed without any warranty expressed or implied. Implementation or use of any information contained herein is the reader's sole responsibility. While the information has been reviewed for accuracy, there is no guarantee that the same or similar results may be achieved in all environments. Technical inaccuracies may result from printing errors and/or new developments in the industry.

Corporate Address: 1105 Media, Inc., 9201 Oakdale Ave., Ste 101, Chatsworth, CA 91311, www.1105media.com

Media Kits: Direct your Media Kit requests to Matt Morollo, VP Publishing, 508-532-1418 (phone), 508-875-6622 (fax), mmorollo@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

All customer service inquiries should be sent to MSDNmag@1105service.com or call 847-763-9560.

Microsoft

BPA WORLDWIDE
Printed in the USA

LEADTOOLS
PDF
OCR

EDITOR'S NOTE

MICHAEL DESMOND

Survey Says

Every couple years we conduct a survey of *MSDN Magazine* subscribers, in an effort to understand the technologies, issues and topics that are of greatest interest to our readers. The results always make for fascinating reading, as we get a glimpse into the tools our readers use and the directions they intend to take with them.

Our 2013 reader survey came during what can only be called an interesting juncture in the evolution of the Microsoft developer ecosystem. Between our 2011 and 2013 surveys, Microsoft released a little thing called Windows 8 and the Windows Runtime. You might have heard of them. The company also deprecated Silverlight, released key improvements to ASP.NET, and launched not one, but two, new updates of the Visual Studio IDE. Platforms such as Windows Azure and Windows Phone continued to advance rapidly, creating increasingly mature and compelling targets for application development.

So it's no surprise that our surveys in 2011 and 2013 have produced an informative snapshot of how these changes are impacting the planning and thinking of *MSDN Magazine* readers.

One thing that hasn't changed: *MSDN Magazine* readers are accomplished. About 23 percent of survey respondents report having worked 25 years or more in the development field, while 55 percent report working between 10 and 25 years. That's nearly 80 percent of *MSDN Magazine* readers with more than a decade of hands-on experience in the field. What's more, 83 percent of readers report being actively involved in programming, versus just 17 percent who are not coding on a daily basis.

The use of tools and languages among *MSDN Magazine* subscribers continues to evolve. In 2011, 61 percent of respondents reported working in C# as their primary language. Two years later in 2013, that figure stood at 65.5 percent. Visual Basic .NET, on the other hand, declined in usage as a primary programming language, from 17 percent of respondents in 2011 to about 12 percent in 2013. C++ usage declined as well over the two year period, from 10 percent to a little more than 6 percent.

We also asked readers what languages and tools are in use in their organizations, and found some interesting trends. Employment of languages such as C#, C++ and Java within companies remained largely stable at organizations between the 2011 and 2013 surveys, but usage of both Visual Basic .NET and Visual Basic 6 dipped. In 2013, 41 percent of respondents reported their companies use Visual Basic .NET, down from 44 percent in 2011. Similarly, Visual Basic 6 use has declined from 21 percent of companies to 17 percent over the two-year span.

JavaScript earns a nod here, as we added the language to those tracked beginning with the 2013 survey. Reporting that their companies employ JavaScript were 67.4 percent of respondents, placing it second only to C# (86.5 percent) among languages deployed within reader organizations. I look forward to our next reader survey, and seeing how the arrival of the Windows Runtime, with its native support for both C++ and JavaScript, impacts uptake of these languages going forward.

No surprise, the vast majority of *MSDN Magazine* readers live and work in Visual Studio, and pickup of the latest versions of the IDE remains prompt. In 2011, 79 percent of readers reported that Visual Studio 2010 was deployed at their companies, followed by Visual Studio .NET 2008 at 64 percent. Two years later, the most widely deployed Visual Studio version was Visual Studio 2012 (68.4 percent), followed by Visual Studio 2010 (58.4 percent) and the just-released Visual Studio 2013 (41.6 percent).

When we asked readers which Microsoft technologies they currently use or plan to use within the next 12 months, we weren't surprised to see technologies such as Visual Studio and the Microsoft .NET Framework produce response rates north of 90 percent. We also weren't surprised to see planned adoption of Silverlight crater, from 43 percent in 2011 to just 16 percent of respondents in 2013. Emerging technologies and platforms, led by Windows Azure (28.4 percent), Windows Phone (21.3 percent) and the Windows Runtime (13.4 percent) have all gained ground.

What will the *MSDN Magazine* readership look like in 2015, 2017 or 2019? It's an open question. Languages like JavaScript certainly figure to grow in importance. In 2013 just 1 percent of *MSDN Magazine* readers identified JavaScript as their primary language—the same percentage that singled out Visual Basic 6 as their main language. I can only guess where that number might sit two or four years from now.

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

WINDOWS WITH C++

KENNY KERR

Using Databases on Windows Azure

Microsoft has a long history of confusing developers with a dizzying array of data access technologies. There was a time when it seemed as if every release of Windows, SQL Server or Visual Studio ushered in a new data access API. Somewhere along the line—around 1996, I believe—Microsoft applied its usual enthusiasm to move developers from ODBC to OLE DB.

ODBC stands for Open Database Connectivity and was the old standard for accessing database management systems. OLE DB stands for … wait for it … Object Linking and Embedding Database and was the new and universal data access utopia. But this name is entirely misleading, which I'll discuss in a moment.

I still remember reading Don Box's column in the July 1999 issue of *MSDN Magazine* (then called the *Microsoft Systems Journal*) where he described the motivation and inspiration leading to OLE DB. I remember thinking at the time that this was a lot more complicated than ODBC but that it sure was a lot more extensible. The reason the name OLE DB is so misleading is that it has nothing to do with OLE and it isn't specifically for databases. It was truly designed as a universal data access model for *all* data—relational or otherwise—such as text, XML, databases, search engines, you name it. OLE DB debuted when COM was all the rage on the Windows platform, so its COM-heavy API and natural extensibility appealed to many developers.

Fortunately, modern C++ comes to the rescue and can make programming ODBC a breeze.

Still, as a relational database API, it never quite achieved the raw performance of ODBC. Subsequent data access technologies, such as those from the Microsoft .NET Framework, dropped all but the database access features so the dream of universal data access began to fizzle out. Then in August 2011 the SQL Server team, the biggest proponents of OLE DB, made the stunning announcement that "Microsoft is aligning with ODBC for native relational data access" (bit.ly/1dsYgTD). They declared the marketplace was moving away from OLE DB and toward ODBC. So it's back to ODBC as you and I figure out how to access SQL Server for the next generation of native C++ applications.

The good news is that ODBC is relatively simple. It's also extremely fast. It was often claimed that OLE DB outperformed ODBC, but this was rarely the case. The bad news is that ODBC is an old C-style API that few developers remember—or ever learned—how to use. Fortunately, modern C++ comes to the rescue and can make programming ODBC a breeze. If you want to access databases on Windows Azure with C++, then you need to embrace ODBC. Let's take a look.

Figure 1 **An ODBC Traits Class**

```
template <SQLSMALLINT T>
struct sql_traits
{
  using pointer = SQLHANDLE;

  static auto invalid() noexcept -> pointer
  {
    return nullptr;
  }

  static auto close(pointer value) noexcept -> void
  {
    VERIFY_(SQL_SUCCESS, SQLFreeHandle(T, value));
  }
};
```

Like so many other C-style APIs, ODBC is modeled around a set of handles representing objects. So I'll again use my trusty unique_handle class template that I've written about and used in numerous columns. You can get a copy of handle.h from dx.codeplex.com and follow along. ODBC handles are, however, a little dumb. The ODBC API needs to be told the type of each handle as it's used, both in creating and freeing a handle (and its underlying object).

A handle type is expressed with a SQLSMALLINT, which is just a short int value. Instead of defining a unique_handle traits class for each type of object that ODBC defines, I'm going to make the traits class itself a template. **Figure 1** shows what this might look like.

The traits class close method in **Figure 1** is where you can begin to see how you need to tell ODBC the type of each handle when used with some of the generic ODBC functions. Because I'm using the latest preview build of the Visual C++ compiler (the November 2013 CTP as of this writing) I'm able to replace the deprecated throw exception specification with the noexcept specifier, opening the door for the compiler to generate more optimal code in some cases. Unfortunately, although this compiler also provides the ability to deduce the return type for auto functions, it includes a bug that prevents it from doing so for member functions of class templates. Of course, because the traits class is itself a class template, a template alias comes in handy:

```
template <SQLSMALLINT T>
using sql_handle = unique_handle<sql_traits<T>>;
```

Now I can define type aliases for the various ODBC objects, such as the environment and statement objects:

```
using environment = sql_handle<SQL_HANDLE_ENV>;
using statement = sql_handle<SQL_HANDLE_STMT>;
```

I'll also define one for connections, although I'll use a more specific name:

```
using connection_handle = sql_handle<SQL_HANDLE_DBC>;
```

The reason for this is that connections require a bit more work to clean up in all cases. While environment and statement objects don't need much more than this, connection objects really need a connection class to reliably deal with connectivity. Before I can deal with that, I need to create an environment.

The generic SQLAllocHandle function creates various objects. Here, again, you see the separation of the object—or at least the handle—and its type. Rather than duplicating this code all over the place, I'll again use a function template to bring the type information back together. Here's a function template for the generic ODBC SQLAllocHandle function:

```
template <typename T>
auto sql_allocate_handle(SQLSMALLINT const type,
                         SQLHANDLE input)
{
  auto h = T {};

  auto const r = SQLAllocHandle(type,
                                input,
                                h.get_address_of());

  // TODO: check result here ...

  return h;
}
```

Of course, this is still just as generic as the ODBC function, but it exposes the genericity in a C++-friendly manner. I'll get back to error handling in a moment. Because this function template will allocate a handle of a given type and return a handle wrapper, I can simply use one of the type aliases I defined earlier. For an environment, I might do this:

```
auto e = sql_allocate_handle<environment>(SQL_HANDLE_ENV, nullptr);
```

The input handle or parent handle, the second parameter, provides an optional parent handle for some logical containment. An environment doesn't have a parent but instead acts as the parent for connection objects. Unfortunately, it takes a bit more effort to create an environment. ODBC requires that I tell it which version of ODBC I'm expecting. I do that by setting an environment attribute with the SQLSetEnvAttr function. Here's what this might look like when wrapped up in a handy helper function:

```
auto create_environment()
{
  auto e = sql_allocate_handle<environment>(SQL_HANDLE_ENV, nullptr);

  auto const r = SQLSetEnvAttr(e.get(),
    SQL_ATTR_ODBC_VERSION,
    reinterpret_cast<SQLPOINTER>(SQL_OV_ODBC3_80),
    SQL_INTEGER);

  // TODO: check result here ...

  return e;
}
```

Figure 2 **Retrieving a SQL Integer Value**

```
auto get_int(statement const & s,
             short const column)
{
  auto value = int {};

  auto const r = SQLGetData(s.get(),
                            column,
                            SQL_C_SLONG,
                            &value,
                            0,
                            nullptr);

  // TODO: check result here ...

  return value;
}
```

At this point I'm ready to create a connection, which, fortunately, is quite simple:

```
auto create_connection(environment const & e)
{
  return sql_allocate_handle<connection_handle>(
    SQL_HANDLE_DBC, e.get());
}
```

Connections are created in the context of an environment. Here you can see that I use the environment as the parent of the connection. I still need to actually make a connection, and that's the job of the SQLDriverConnect function, some of whose parameters may be ignored:

```
auto connect(connection_handle const & c,
             wchar_t const * connection_string)
{
  auto const r = SQLDriverConnect(c.get(), nullptr,
    const_cast<wchar_t *>(connection_string),
    SQL_NTS, nullptr, 0, nullptr,
    SQL_DRIVER_NOPROMPT);

  // TODO: check result here ...
}
```

Notably, the SQL_NTS constant just tells the function the preceding connection string is null terminated. You could, instead, opt to provide the length explicitly. The final SQL_DRIVER_NOPROMPT constant indicates whether to prompt the user if more information is required to establish a connection. In this case, I'm saying "no" to prompts.

But as I alluded earlier, gracefully closing a connection is a little more involved. The trouble is that while the SQLFreeHandle function is used to free the connection handle, it assumes the connection is closed and won't automatically close an open connection.

What I need is a connection class that tracks the connection's connectivity. Something like this:

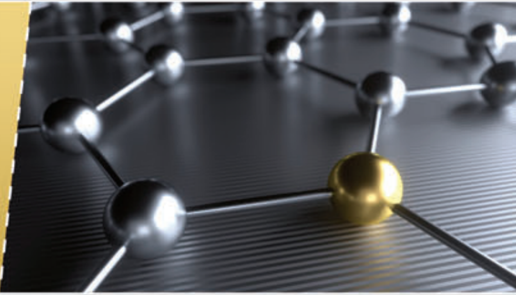
```
class connection
{
  connection_handle m_handle;
  bool m_connected { false };

public:

  connection(environment const & e) :
    m_handle { create_connection(e) }
  {}

  connection(connection &&) = default;

  // ...
};
```

I can now add a connect method to my class using the previously defined non-member connect function and update the connected state accordingly:

```
auto connect(wchar_t const * connection_string)
{
  ASSERT(!m_connected);

  ::connect(m_handle, connection_string);

  m_connected = true;
}
```

Empower Your Customers

New! v5.0



Create & Edit PDFs in .Net - ActiveX - WinRT

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

Complete Suite of Accurate PDF Components

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

Advanced HTML to PDF & XAML

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

High Performance PDF Printer For Desktops and Servers

- Our high-performance printer driver optimized for Web, Application and Print Servers. Print to PDF in a fraction of the time needed with other tools. WHQL tested for Windows 32 and 64-bit including Windows Server 2012 and Windows 8
- Standard PDF features included with a number of unique features. Interface with any .Net or ActiveX programming language
- Easy licensing and deployment to fit system administrator's requirements

CERTIFIED FOR Windows Server® 2008

Windows®7

Windows 8

Windows Server 2012 Certified

AMYUNI

All development tools available at

www.amyuni.com

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

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

The connect method asserts the connection is not open to begin with and keeps track of the fact that the connection is open at the end. The connection class destructor can then automatically disconnect as needed:

```
~connection()
{
  if (m_connected)
  {
    VERIFY_(SQL_SUCCESS, SQLDisconnect(m_handle.get()));
  }
}
```

This will ensure the connection is disconnected prior to the member handle destructor being called to free the connection handle itself. I can now create an ODBC environment and establish a connection correctly and efficiently with just a few lines of code:

```
auto main()
{
  auto e = create_environment();
  auto c = connection { e };

  c.connect(L"Driver=SQL Server Native Client 11.0;Server=...");
}
```

What about statements? The sql_allocate_handle function template again comes in handy, and I'll just add another method to my connection class:

```
auto create_statement()
{
  return sql_allocate_handle<statement>(SQL_HANDLE_STMT,
                                        m_handle.get());
}
```

Statements are created in the context of a connection. Here you can see how the connection is the parent for the statement object. Back in my main function, I can create a statement object quite simply:

```
auto s = c.create_statement();
```

ODBC provides a relatively simple function for executing SQL statements, but I'll again wrap it up for convenience:

```
auto execute(statement const & s,
             wchar_t const * text)
{
  auto const r = SQLExecDirect(s.get(),
                               const_cast<wchar_t *>(text),
                               SQL_NTS);

  // TODO: check result here ...
}
```

ODBC is an extremely old C-style API so it doesn't use const, not even conditionally for C++ compilers. Here, I need to cast away the "const-ness" so the caller is shielded from this const-ignorance. Back in my main function, I can execute SQL statements quite simply:

```
execute(s, L"create table Hens ( ... )");
```

But what if I execute a SQL statement that returns a result set? What if I execute something like this:

```
execute(s, L"select Name from Hens where Id = 123");
```

In that case, the statement effectively becomes a cursor and I need to fetch the results, if any, one at a time. That's the role of the SQLFetch function. I might want to know whether a hen with the given Id exists:

```
if (SQL_SUCCESS == SQLFetch(s.get()))
{
  // ...
}
```

On the other hand, I might execute a SQL statement that returns multiple rows:

```
execute(s, L"select Id, Name from Hens order by Id desc");
```

In that case, I can simply call the SQLFetch function in a loop:

```
while (SQL_SUCCESS == SQLFetch(s.get()))
{
  // ...
}
```

Getting the individual column values is what the SQLGetData function is for. This is another generic function, and you need to precisely describe the information you expect as well as the buffer where you expect it to copy the resulting value. Retrieving a fixed-size value is relatively straightforward. **Figure 2** shows a simple function to retrieve a SQL int value.

The first parameter in SQLGetData is the statement handle, the second is the one-based column index, the third is the ODBC type for a SQL int and the fourth is the address of the buffer that will receive the value. The second-to-last parameter is ignored because this is a fixed-size data type. For other data types, this would indicate the size of the buffer on input. The final parameter provides the actual length or size of the data copied into the buffer. Again, this isn't used for fixed-size data types, but this parameter may also be used to return status information such as whether the value was null. Retrieving a string value is only slightly more complicated. **Figure 3** shows a class template that will copy the value into a local array.

Notice how in this case I need to tell the SQLGetData function the actual size of the buffer to receive the value, and I need to do so in bytes, not characters. If I queried for the name of a particular hen and the Name column holds a maximum of 100 characters, I might use the get_string function, as follows:

```
if (SQL_SUCCESS == SQLFetch(s.get()))
{
  wchar_t name[101];
  get_string(s, 1, name);
  TRACE(L"Hen's name is %s\n", name);
}
```

Finally, while I can reuse a connection object to execute multiple statements, once the statement object represents a cursor,

Figure 3 **Retrieving a SQL String Value**

```
template <unsigned Count>
auto get_string(statement const & s,
                short const column,
                wchar_t (&value)[Count])
{
  auto const r = SQLGetData(s.get(),
                            column,
                            SQL_C_WCHAR,
                            value,
                            Count * sizeof(wchar_t),
                            nullptr);

  sql_check(r, SQL_HANDLE_STMT, s.get());
}
```

Figure 4 **Retrieving Diagnostic Error Information**

```
auto native_error = long {};
wchar_t state[6];
wchar_t message[1024];

auto record = short {};

while (SQL_SUCCESS == SQLGetDiagRec(type,
                                    handle,
                                    ++record,
                                    state,
                                    &native_error,
                                    message,
                                    _countof(message),
                                    nullptr))
{
  // ...
}
```

I need to be sure to close the cursor before executing any subsequent statements:

```
VERIFY_(SQL_SUCCESS, SQLCloseCursor(s.get()));
```

Ironically, this isn't a resource management issue. Unlike the challenges with open connections, the SQLFreeHandle function doesn't care if the statement has an open cursor.

I've avoided talking about error handling up until now because it's a complex topic in its own right. ODBC functions return error codes, and it's your responsibility to check the value of these return codes to determine whether the operation succeeded. Usually the functions will return the SQL_SUCCESS constant indicating success, but they can also return the SQL_SUCCESS_WITH_INFO constant. The latter is equally successful but implies further diagnostic information is available if you wish to retrieve it. Typically, only in debug builds do I retrieve the diagnostic information when the SQL_SUCCESS_WITH_INFO constant is returned. This way I can gather as much information as possible in development and not waste cycles in production. Of course, I always gather this information when an error code is actually returned. Regardless of the cause, the process by which the diagnostic information is retrieved is the same.

ODBC provides diagnostic information as a result set and you can retrieve the rows one at a time with the SQLGetDiagRec function and a one-based row index. Just make sure to call it with the handle of the object that produced the error code in question.

There are three principal bits of information in each row: a native error code specific to the ODBC data source or driver; a short, cryptic, five-character state code that defines the class of error to which this record might refer; and a longer textual description of the diagnostic record. Given the necessary buffers, I can simply call the SQLGetDiagRec function in a loop to retrieve them all, as shown in **Figure 4**.

Windows Azure along with SQL Server provides an amazingly simple way to get started with hosted databases. This is particularly compelling as the SQL Server database engine is the same one C++ developers have known and used for years. While OLE DB has been scrapped, ODBC is more than up to the task and, in fact, is simpler and faster than OLE DB ever was. Of course, it takes a bit of help from C++ to make it all come alive in a coherent way.

Check out my Pluralsight course, "10 Practical Techniques to Power Your Visual C++ Apps" (bit.ly/1fgTifi), for more information about using Visual C++ to access databases on Windows Azure. I provide step-by-step instructions to set up and use database servers and bind columns to simplify the process of fetching rows of data, examples of how to simplify and modernize the error-handling process, and much more. ■

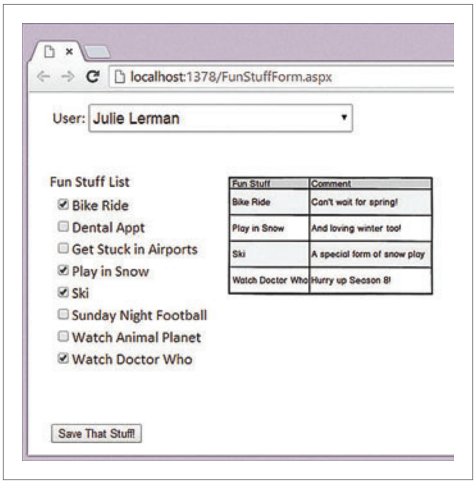
Kenny Kerr *is a computer programmer based in Canada, as well as an author for Pluralsight and a Microsoft MVP. He blogs at kennykerr.ca and you can follow him on Twitter at twitter.com/kennykerr.*

DOC
PPT
XLS
PDF
MSG

Adding New Life to a 10-Year-Old ASP.NET Web Forms App

Legacy code: can't live with it, can't live without it. And the better job you do with an app, the longer it will hang around. My very first ASP.NET Web Forms app has been in use for a little more than 10 years. It's finally getting replaced with a tablet app someone else is writing. However, in the meantime, the client asked me to add a new feature to it that will let the company start collecting right away some of the data the new version will gather.

This isn't a matter of a simple field or two. In the existing app—a complicated time sheet for tracking employee hours—there's a dynamic set of checkboxes defined by a list of tasks. The client maintains that list in a separate application. In the Web app, a user can check any number of items on that list to specify the tasks he performed. The list contains a little more than 100 items and grows slowly over time.



localhost:1378/FunStuffForm.aspx

User: Julie Lerman

Fun Stuff List

☑ Bike Ride
☐ Dental Appt
☐ Get Stuck in Airports
☑ Play in Snow
☑ Ski
☐ Sunday Night Football
☐ Watch Animal Planet
☑ Watch Doctor Who

Fun Stuff	Comment
Bike Ride	Can't wait for spring!
Play in Snow	And loving winter too!
Ski	A special form of snow play
Watch Doctor Who	Hurry up Season 8!

Figure 1 **Starting Point: A Simple ASP.NET Web Form with the Planned Addition**

Now, the client wants to track the number of hours spent on each selected task. The app will be used for only a few more months, so it didn't make sense to invest a lot into it, but I had two important goals regarding the change:

1. Make it really easy for the user to enter the hours, which meant not having to click any extra buttons or cause postbacks.
2. Add the feature to the code in the least invasive way possible. While it would be tempting to overhaul the 10-year-old app with more modern tools, I wanted to add the new logic in a way that wouldn't impact existing (working) code, including data access and the database.

I spent some time considering my options. Goal No. 2 meant leaving the CheckBoxList intact. I decided to contain the hours in a separate grid, but Goal No. 1 meant not using the ASP.NET GridView control (thank goodness). I decided to use a table and JavaScript for retrieving and persisting the task-hours data and I explored a few ways to achieve this. AJAX PageMethods to call the codebehind couldn't be used because my page was retrieved using a Server.Transfer from another page. Inline calls, such as <%MyCodeBehindMethod()%>, worked until I had to do some complex data validation (too difficult to accomplish in JavaScript) that required a mix of client- and server-side objects. The situation also started getting ugly with the need to make everything touched by the inline call static. So I was failing at "least invasive."

Finally, I realized I should really aim to keep the new logic totally separate and put it into a WebAPI that would be easy to access from JavaScript. This would help keep a clean separation between the new logic and the old.

Still, I had challenges. My prior experience with Web API was to create a new MVC project. I started with that, but calling methods in the Web API from the existing app was causing Cross Origin Resource Sharing (CORS) issues that defied every pattern I could find for avoiding CORS. Finally, I discovered an article by Mike Wasson about adding a Web API directly into a Web Forms project (bit.ly/1jNZKzI) and I was on my way—though I had many bridges yet to cross. I won't make you relive my pain as I bashed, thrashed and fumbled my way to success. Instead, I'll walk you through the solution I ultimately reached.

Rather than present my client's real application to demonstrate how I brought the new functionality into the old app, I'll use a sample that tracks user preferences via comments about things they like to do: fun stuff. I'll forgo the list of 100-plus items here; the form shows only a short CheckBoxList, as well as the additional work for data validation. And instead of tracking hours, I'll track user comments.

Once I committed to the Web API, adding the validation method wasn't a challenge at all. Because I was creating a new sample, I used the Microsoft .NET Framework 4.5 and Entity Framework 6 (EF) instead of .NET Framework 2.0 and raw ADO.NET. **Figure 1** shows the starting point of the sample application: an ASP.NET Web Form with a user name and an editable CheckBoxList of possible activities. This is the page to which I'll add the ability to track comments for each checked item as shown by the sketched-in grid.

Step 1: Add the New Class

I needed a class to store the new comments. I concluded that, given my data, it made the most sense to use a key composed of

Code download available at msdn.microsoft.com/magazine/msdnmag0414.

Figure 2 **BreezeController Web API**

```
namespace April2014SampleWebForms{
[BreezeController]
public class BreezeController: ApiController  {
  [HttpGet]
  public IQueryable<FunStuffComment> Comments(int userId = 0)
    if (userId == 0){ // New user
      return new List<FunStuffComment>().AsQueryable();
    }
      return new List<FunStuffComment>{
        new FunStuffComment{FunStuffName = "Bike Ride",
          Comment = "Can't wait for spring!",FunStuffId = 1,UserId = 1},
        new FunStuffComment{FunStuffName = "Play in Snow",
          Comment = "Will we ever get snow?",FunStuffId = 2,UserId = 1},
        new FunStuffComment{FunStuffName = "Ski",
          Comment = "Also depends on that snow",FunStuffId = 3,UserId = 1}
      }.AsQueryable();    }
  }
}
```

UserId and FunStuffId to determine to which user and fun activity the comment would attach:

```
namespace DomainTypes{
  public class FunStuffComment{
    [Key, Column(Order = 0)]
    public int UserId { get; set; }
    [Key, Column(Order = 1)]
    public int FunStuffId { get; set; }
    public string FunStuffName { get; set; }
    public string Comment { get; set; }
  }
}
```

Because I planned to use EF for persisting the data, I needed to specify the properties that would become my composite key. In EF, the trick for mapping composite keys is to add the Column Order attribute along with the Key attribute. I also want to point out the FunStuffName property. Even though I could cross-reference my FunStuff table to get the name of a particular entry, I found it easier to simply surface FunStuffName in this class. It might seem redundant, but keep in mind my goal to avoid messing with the existing logic.

Step 2: Adding Web API to the Web Forms-Based Project

Thanks to Wasson's article, I learned I could add a Web API controller directly into the existing project. Just right-click the project in Solution Explorer and you'll see Web API Controller Class as an option under the Add context menu. The controller that's created is designed to work with MVC, so the first order of business is to remove all of the methods and add in my Comments method for retrieving existing comments for a particular user. Because I'll be using the Breeze JavaScript library and have already installed it into my project using NuGet, I use Breeze naming conventions for my Web API Controller class, as you can see in **Figure 2**. I haven't hooked the Comments into my data access yet, so I'll begin by returning some in-memory data.

Figure 3 **HTML Table Set Up for Binding with Knockout**

```
<table id="comments">
  <thead>
    <tr>
      <th></th>
      <th></th>
      <th>Fun Stuff</th>
      <th>Comment</th>
    </tr>
  </thead>
  <tbody data-bind="foreach: comments">
    <tr>
      <td style="visibility: hidden" data-bind="text: UserId"></td>
      <td style="visibility: hidden" data-bind="text: FunStuffId"></td>
      <td data-bind="text: FunStuffName"></td>
      <td><input data-bind="value: Comment" /></td>
    </tr>
  </tbody>
</table>
```

Wasson's article guides you to add routing to the global.asax file. But adding Breeze via NuGet creates a .config file with the appropriate routing already defined. That's why I'm using the Breeze recommended naming in the controller in **Figure 2**.

Now I can call the Comments method easily from the client side of my FunStuffForm. I like to test my Web API in a browser to make sure things are working, which you can do by running the app and then browsing to http://localhost:1378/breeze/Breeze/Comments?UserId=1. Be sure to use the correct host:port your app is using.

Step 3: Adding Client-Side Data Binding

But I'm not done yet. I need to do something with that data, so I looked back to my previous columns on Knockout.js (msdn.microsoft.com/magazine/jj133816 for JavaScript data binding) and Breeze (msdn.microsoft.com/magazine/jj863129, which makes the data binding even simpler). Breeze automatically transforms the results of my Web API into bindable objects that Knockout (and other APIs) can use directly, eliminating the need to create additional view models and mapping logic. Adding the data binding is the most intensive part of the conversion, made worse by my still very limited JavaScript and jQuery skills. But I persevered—and also became a semi-pro at JavaScript debugging in Chrome along the way. Most of the new code is in a separate JavaScript file that's tied to my original Web Form page, FunStuffForm.aspx.

When I was nearly done with this article, someone pointed out that Knockout is now a bit dated ("It's so 2012," he said), and many JavaScript developers are using the simpler and richer frameworks such as AngularJS or DurandalJS instead. That's a lesson for me to learn another day. I'm sure my 10-year-old app won't mind a 2-year-old tool. But I'll definitely be taking a look at these tools in a future column.

In my Web Form, I defined a table named comments with columns populated by fields of the data I'll be binding to it with Knockout (see **Figure 3**). I'm also binding the UserId and FunStuffId fields, which I'll need later, but keeping them hidden.

The first chunk of logic in the JavaScript file that I called FunStuff.js is what's known as a *ready function* and it will run as soon as the rendered document is ready. In my function, I define the viewModel type, shown in **Figure 4**, whose comments property I'll use to bind to the comments table in my Web Form.

The ready function also specifies some startup code:

- **serviceName** defines the Web API uri
- **vm** is a short alias for viewModel
- **manager** sets up the Breeze EntityManager for the Web API
- **getComments** is a method that calls the API and returns data
- **ko.applyBinding** is a Knockout method to bind the viewModel to the comments tables

Figure 4 **Beginning of FunStuff.js**

```
var viewModel;
$(function() {
  viewModel = {
    comments: ko.observableArray(),
    addRange: addRange,
    add: add,
    remove: remove,
    exists: exists,
    errorMessage: ko.observable(""),
  };

  var serviceName = 'breeze/Comments';
  var vm = viewModel;
  var manager = new breeze.EntityManager(serviceName);
  getComments();
  ko.applyBindings(viewModel, document.getElementById('comments'));

 // Other functions follow
});
```

Figure 5 **Querying Data Through the Web API Using Breeze**

```
function getComments () {
  var query = breeze.EntityQuery.from("Comments")
    .withParameters({ UserId: document.getElementById('hiddenId').value });
  return manager.executeQuery(query)
    .then(saveSucceeded).fail(failed);
}

function saveSucceeded (data) {
  var count = data.results.length;
  log("Retrieved Comments: " + count);
  if (!count) {
    log("No Comments");
    return;
  }
  vm.comments(data.results);
}
function failed(error) {
  vm.errorMessage(error);
}
```

Notice that I've declared viewModel outside of the function. I'll need access to it from a script in the .aspx page later, so it had to be scoped for external visibility.

The most important property in viewModel is an observableArray named *comments*. Knockout will keep track of what's in the array and update the bound table when the array changes. The other properties just expose additional functions I've defined below this startup code through the viewModel.

Let's start with the getComments function shown in **Figure 5**.

In the getComments function, I use Breeze to execute my Web API method, Comments, passing in the current UserId from a hidden field on the Web page. Remember I've already defined the uri of Breeze and Comments in the manager variable. If the query succeeds, the saveSucceeded function runs, logging some info on the screen and pushing the results of the query into the comments property of the viewModel. On my laptop, I can see the empty table before the asynchronous task is complete and then suddenly the table is populated with the results (see **Figure 6**). And remember, this is all happening on the client side. No postbacks are occurring so it's a fluid experience for the user.

Step 4: Reacting to Boxes Being Checked and Unchecked

The next challenge was to make that list respond to the user's selections from the Fun Stuff List. When an item is checked, it needs to be added or removed from the viewModel.comments array and the bound table depending on whether the user is adding or removing a checkmark. The logic for updating the array is in the JavaScript file, but the logic for alerting the model about the action resides in a script in the .aspx. It's possible to bind functions such as a checkbox onclick to Knockout, but I didn't take that route.

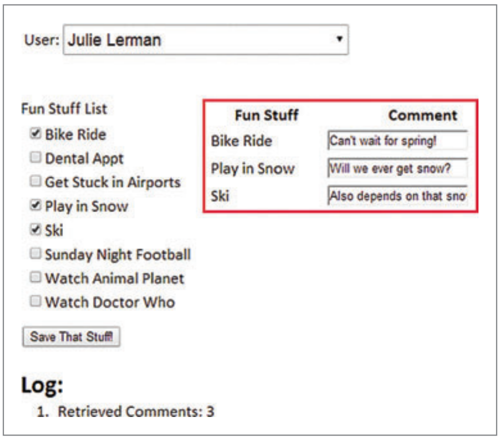
In the markup of the .aspx form, I added the following method to the page header section:

```
$("#checkBoxes").click(function(event) {
  var id = $(event.target)[0].value;
  if (event.target.nodeName == "INPUT") {
    var name = $(event.target)[0].parentElement.textContent;
    // alert('check!' + 'id:' + id + ' text:' + name);
    viewModel.updateCommentsList(id, name);
  }
});
```

This is possible thanks to the fact that I have a div named checkBoxes surrounding all of the dynamically generated CheckBox controls. I use jQuery to grab the value of the CheckBox that's triggering the event and the name in the related label. Then I pass those on to the updateCommentsList method of my viewModel. The alert is just for testing that I had the function wired properly.

Now let's take a look at the updateCommentsList and related functions in my JavaScript file. A user might check or uncheck an item, so it needs to be either added or removed. Rather than worry about the state of the checkbox, in my exists method I just let the Knockout utils function help me see if the item is already in the array of comments. If it is, I need to remove it. Because Breeze is tracking changes, I remove it from the observableArray but tell the Breeze change tracker to consider it deleted. This does two things. First, when I save, Breeze sends a DELETE command to the database (via EF in my case). But if the item is checked again and needs to be added back into the observableArray, Breeze simply restores it in the change tracker. Otherwise, because I'm using a composite key for the identity of comments, having both a new item and a deleted item with the same identity would create a conflict. Notice that while Knockout responds to the push method for adding items, I must notify it that the array has mutated in order for it to respond to removing an item. Again, because of the data binding, the table changes dynamically as checkboxes are checked and unchecked.

Notice that when I create a new item, I'm grabbing the user's userId from the hidden field in the form's markup. In the original version of the form's Page_Load, I set this value after grabbing the user.



User: Julie Lerman

Fun Stuff List
- ☑ Bike Ride
- □ Dental Appt
- □ Get Stuck in Airports
- ☑ Play in Snow
- ☑ Ski
- □ Sunday Night Football
- □ Watch Animal Planet
- □ Watch Doctor Who

Fun Stuff	Comment
Bike Ride	Can't wait for spring!
Play in Snow	Will we ever get snow?
Ski	Also depends on that sno

Log:
1. Retrieved Comments: 3

Figure 6 **Comments Retrieved from Web API and Bound with the Help of Knockout.js**

REMIUM	REGION	STATUS
$572	Northeast	Active
$624	Northeast	Lapsed
$482	West	Active
$389	Northwest	Active
$874	Southwest	Lapsed
$291	West	Active
$288	Midwest	Active

Built for Today… Ready for Tomorrow

Touch-Enabled .NET Controls by DevExpress.

Create solutions your customers expect today and leverage your existing skillsets to build next generation applications for tomorrow. DevExpress Controls are built to emulate the UI experiences at the heart of Microsoft Office AND to enable touch-first experiences for next-generation devices like iPad and Surface.

Your next great app starts here.
DevExpress.com/Touch

DevExpress®

By tying the UserId and FunStuffId to each item in the comments, I can store all of the necessary data along with the comments to associate them with the correct user and item.

With oncheck wired up and the comments observableArray modified in response, I can see that, for example, toggling the Watch Doctor Who checkbox causes the Watch Doctor Who row to display or disappear based on the state of the checkbox.

Step 5: Saving Comments

My page already has a Save feature for saving the checkboxes marked true, but now I want to save the comments at the same time using another Web API method. The existing save method executes when the page posts back in response to the SaveThatStuff button click. Its logic is in the page codebehind. I can actually make a client-side call to save the comments prior to the server-side call using the same button click. I knew this was possible with Web Forms using an old-school onClientClick attribute, but in the timesheet application I was modifying, I also had to perform a validation that would determine if the task hours and time sheet were ready to be saved. If the validation failed, not only did I have to forget about the Web API save, but I had to prevent the postback and server-side save method from executing as well. I was having a hard time working this out using onClientClick, which encouraged me to modernize again with jQuery. In the same way I can respond to the CheckBox clicks in the client, I can have a client-side response to btnSave being clicked. And it will happen prior to the postback and server-side response. So I get to have both events on one click of the button, like so:

```
$("#btnSave").click(function(event) {
  validationResult = viewModel.validate();
  if (validationResult == false) {
    alert("validation failed");
    event.preventDefault();
  } else {
    viewModel.save();
  }
});
```

I have a stub validation method in the sample that always returns true, though I tested to be sure things behave properly if it returns false. In that case, I use the JavaScript event.preventDefault to stop further processing. Not only will I not save the comments, but the postback and server-side save will not occur. Otherwise, I call viewModel.save and the page continues with the button's server-side behavior, saving the user's FunStuff choices. My saveComments function is called by viewModel.save, which asks the Breeze entityManager to execute a saveChanges:

```
function saveComments() {
  manager.saveChanges()
    .then(saveSucceeded)
    .fail(failed);
}
```

This in turn finds my controller SaveChanges method and executes it:

```
[HttpPost]
  public SaveResult SaveChanges(JObject saveBundle)
  {
    return _contextProvider.SaveChanges(saveBundle);
  }
```

For this to work, I added Comments into the EF6 data layer and then switched the Comments controller method to execute a query against the database using the Breeze server-side component (which makes a call to my EF6 data layer). So the data returned to the client will be data from the database, which SaveChanges can then save back to the database. You can see this in the download sample, which uses EF6 and Code First and will create and seed a sample database.

Figure 7 **JavaScript for Updating the Comments List in Response to a User Clicking on the Checkboxes**

```
function updateCommentsList(selectedValue, selectedText) {
  if (exists(selectedValue)) {
    var comment = remove(selectedValue);
    comment.entityAspect.setDeleted();
  } else {
  var deleted = manager.getChanges().filter(function (e) {
    return e.FunStuffId() == selectedValue
  })[0];  // Note: .filter won't work in IE8 or earlier
  var newSelection;
  if (deleted) {
    newSelection = deleted;
    deleted.entityAspect.rejectChanges();
  } else {
    newSelection = manager.createEntity('FunStuffComment', {
      'UserId': document.getElementById('hiddenId').value,
      'FunStuffId': selectedValue,
      'FunStuffName': selectedText,
      'Comment': ""
    });
  }
  viewModel.comments.push(newSelection);    }

  function exists(stuffId) {
    var existingItem = ko.utils.arrayFirst(vm.comments(), function (item) {
      return stuffId == item.FunStuffId();
    });
    return existingItem != null;
  };
  function remove(stuffId) {
    var selected = ko.utils.arrayFirst
    (vm.comments(), function (item) {
    return stuffId == item.FunStuffId;
    });
    ko.utils.arrayRemoveItem(vm.comments(), selected);
    vm.comments.valueHasMutated();
  };
```

JavaScript with a Little Help from My Friends

Working on this project and on the sample built for this article, I wrote more JavaScript than I ever had before. It's not my area of expertise (as I've pointed out frequently in this column), though I was quite proud of what I had accomplished. However, knowing that many readers might be seeing some of these techniques for the first time, I leaned on Ward Bell from IdeaBlade (the creators of Breeze) for an in-depth code review, along with some pair programming to help me clean up some of my Breeze work as well as JavaScript and jQuery. Except perhaps for the now "dated" use of Knockout.js, the sample you can download should provide some good lessons. But, remember, the focus is about enhancing an old Web Forms project with these more modern techniques that make the end-user experience so much more pleasant. ■

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 Microsoft .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 at twitter.com/julielerman and see her Pluralsight courses at juliel.me/PS-Videos.*

THANKS to the following Microsoft technical experts for reviewing this article: *Damian Edwards and Scott Hunter*

Windows Store C++ for C# Developers: Understanding the Domain

Bill Kratochvil

C# developers have a rich development environment that emphasizes productivity over performance. I suspect this statement might be met with some skepticism from C# developers as performance is excellent and exceeds what most of us need to accomplish our goals. Thus, for C# developers, modern C++ may not offer a return on investment as productivity may suffer even with all of the new improvements the new C++ offers. However, the beauty of Windows Store app development is that you can take an agile approach to building reusable components that's productive, using the best of both languages to phase in C++ components where they're most useful.

This article discusses:

- The WinRT type system
- Libraries
- Modern C++
- Setting up Visual Studio to use the Windows Store application C++ project templates

Technologies discussed:

C#, C++, C++/CX, Windows Runtime C++ Template Library, JavaScript

Code download available at:

msdn.microsoft.com/magazine/msdnmag0414

This article is intended for C# developers who want to tap into the power and performance that modern C++, C++/CX (component extensions), and the Windows Runtime C++ Template Library (WRL) offer (all referred to as C++ here for brevity). It's also intended for developers who, like me, plan on leveraging the power of the Windows Azure cloud. Applications built with C++ not only have the potential to extend tablet and phone battery life but should also result in lower cost due to better performance per watt, per transistor and per cycle (see "C++ and Beyond 2011: Herb Sutter – Why C++?" at bit.ly/1gtKQya).

The problem C# developers encounter in attempting to learn C++ is that most of the documentation, blogs and other material available reflect the perspectives of experienced C++ developers, not the C# developer (who generally doesn't want to build a single-project console application). To further complicate matters, C++ is undergoing a renaissance, so much of the current material could be obsolete, not apply or distract you from the newer, more efficient coding techniques. To discern what's valid and what isn't, you need a basic understanding of the domain, indicated by acronyms such as C++, C++/CLI, C++/CX, WRL, STL, ATL, WinRT and so on. Without this understanding you could end up burning valuable cycles learning material that doesn't apply to your objective, which is, most likely, ramping up quickly to build C++ Windows Store applications.

My interest in C++ came about when I hit the C# performance ceiling with a use case that had to process a complex algorithm on more than 200,000 records. My research took me to the webcast, "Taming GPU Compute with C++ AMP" (bit.ly/1dajKE6), which presented a demo application that changed my perception of performance forever. Had I had experience with C++ AMP, and been able to create a proof of concept to present to the team, I suspect they would've gladly accepted a C++ component when they saw an 11-second process reduced to milliseconds. Unfortunately, the magnitude of C++ information (and disinformation) available, coupled with my lack of C++ domain knowledge, had me stumbling into a vast library of information, searching for the shelf that contained the books pertaining to C++ development for Windows Store applications.

C# developers have a rich development environment that emphasizes productivity over performance.

The purpose of this article is not to teach you C++, but to point you to that shelf. This will ensure your learning path is clear, free of muddied waters, and you're not distracted by the blizzard of information that can actually block your progress and cause you unnecessary frustration. After reading this article, you should be better equipped to benefit from *applicable* C++ material, such as the more recent *MSDN Magazine* articles, libraries and blogs about C++ development.

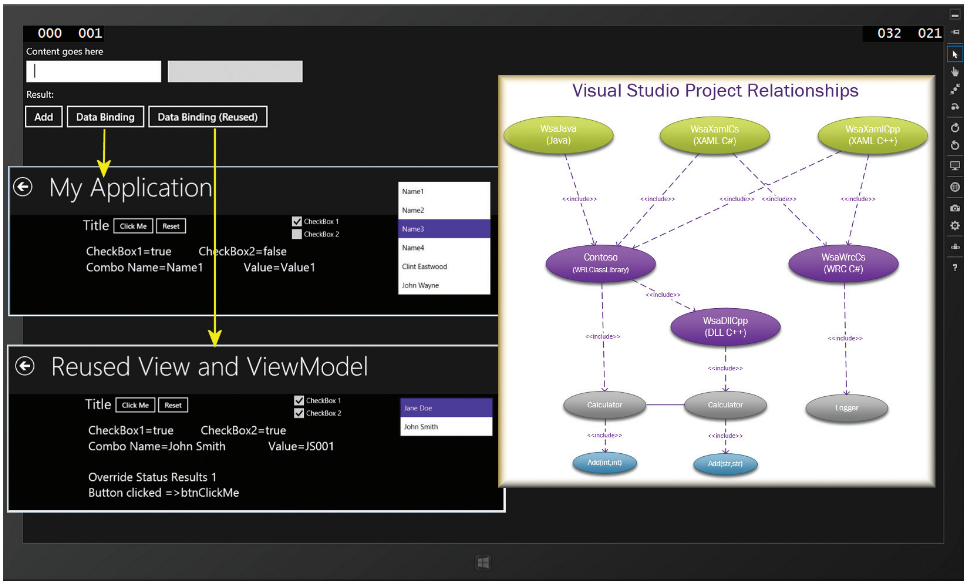
The code download for this article is a Visual Studio solution containing three apps based on different development platforms: C#, C++ and JavaScript. As you'll see, within this solution I abbreviate Windows Store applications as Wsa for my project names (this is not an official abbreviation). The WsaJavaScript application is based on the MSDN tutorial, "Walkthrough: Creating a Basic Windows Runtime Component Using WRL" (bit.ly/1n1gW2). You'll see that I simply added two buttons labeled "Data Binding" and "Data Binding (Reused)" to this tutorial application (as shown in **Figure 1**). Note that both the WsaXamlCpp (C++/CX) and WsaXamlCs (C#) applications in this solution use the same libraries and yield the same exact results.

The C# and C++/CX applications demonstrate just how powerful Windows Store application development really is—you can seamlessly reuse components between platforms, as the right pane of **Figure 1** shows). For example, as you review the code, you'll find both the C# and C++/CX demo applications use the same XAML, which was copied and pasted into each application's respective view (DataBindingView). Likewise, you'll see they use the same presenter (DataBindingPresenter), view model (MainViewModel), data access layer (MockDataDal) and logger (DebugLogger). With Windows Store application development, you can create C# or C++ libraries in either language, so you can step into C++ development at your own pace, and replace less efficient C# algorithms with high-performance C++ AMP code with minimal productivity loss.

Productivity shines because you can recycle all of your useful code. For example, the logger for both applications (see the output window at the bottom of **Figure 2**) is a C# component that resides in the C# WsaWrcCs project. I didn't have to invest time building a C++ logger, which has negligible impact on performance, and could focus on more critical areas. **Figure 2** also reveals there's little difference between the C# and C++/CX code in the DataBindingPresenter class; this class contains most of the application's code as all other logic resides in the shared components (you'll find little code in both application projects).

Besides showing off the power of Windows Store application development, the solution presented here provides a sandbox in which you can start C++ development. Where most tutorials and examples provide code for a console app (having little value in a Windows Store application environment), this solution gives you sample code in the applicable project type in which you're interested—modern C++, WRL or C++/CX. You can see how your C++ code can be easily accessed by the C# application.

While I'm going to discuss the C++ domain topics applicable to Windows Store app development, they won't be covered in great depth here. However, the following should provide a baseline from which you can continue your research, and I'll provide some links so you can drill down into the details, when applicable. The point of this exercise is to help you focus on the key topics you need to understand, effectively filtering out the magnitude of information



CheckBox1=true CheckBox2=false
Combo Name=Name1 Value=Value1

CheckBox1=true CheckBox2=true
Combo Name=John Smith Value=JS001

Override Status Results 1
Button clicked =>btnClickMe

Figure 1 **Three Application Views (Applications Shown in Green, Libraries in Purple, Use Cases in Gray)**

that doesn't apply. I want to acknowledge the Microsoft C++ development team, as they were instrumental in helping me navigate through the massive library of information to find the shelf I needed so I could be productive in my journey to becoming a Microsoft C++ developer.

Windows Runtime (WinRT)

The WinRT type system lets you use a variety of languages to access Windows APIs (that are exposed as WinRT types) and to write apps and components using the same technology. The Windows Runtime is based on an Application Binary Interface (ABI); a standard for connecting components, as well as APIs.

What you see in **Figure 1** is the seamless integration of different development platforms: C#, C++, and C++/CX transparently being used by JavaScript, C# and C++ Windows Store applications. For this to be possible there has to be a standard interface all development languages adhere to so they can use the compiled code. To achieve this, the WRL and WinRT component projects each generate a .winmd file that serves as a cross-language header file. Note that the .winmd file is essentially a CLI metadata file (so it can be viewed with ILSpy).

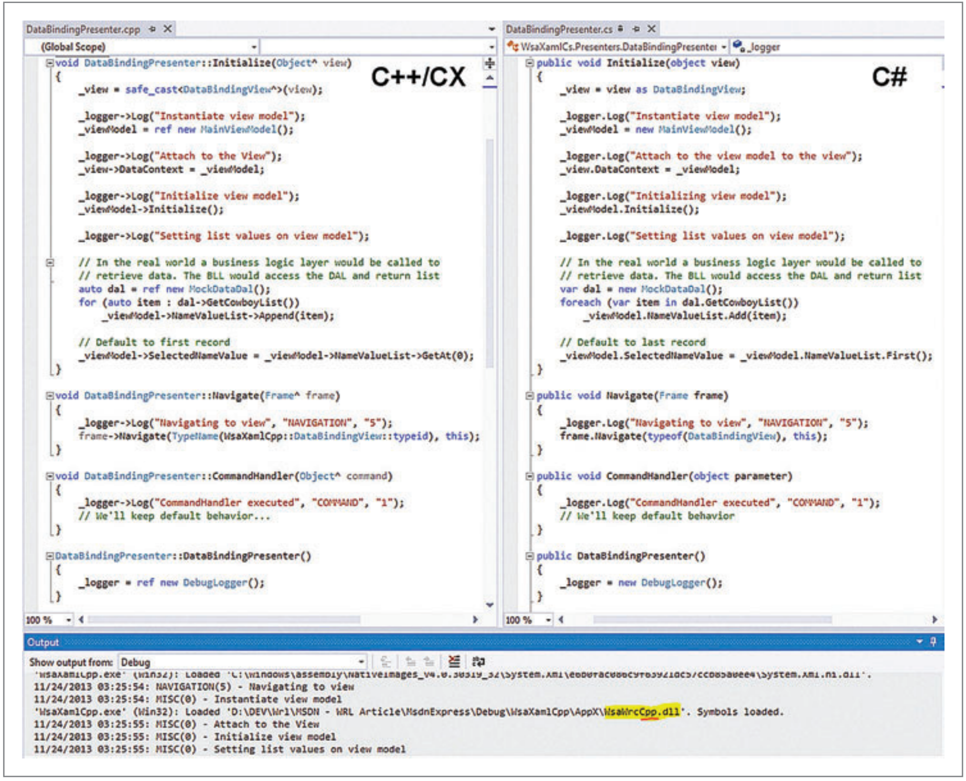
You produce and consume the ABI using C++/CX, WRL, C# and JavaScript (or any other language that adds support for the WinRT ABI). As a C# developer your inclination might be to write all of your code using C++/CX, as it's closest to what you're accustomed to using, but you should minimize the use of WinRT as much as possible, and strictly limit it to the ABI layer. For example, your code should leverage the power of C++ using only C++/CX when necessary to cross the ABI. Consistent with its design and purpose, you'll find that C++/CX contains only a small subset of functions for its components, that is, String will be missing most of the functions to which you're accustomed and may need, as it was not intended to be a stand-alone development language. Understanding the ABI and its constraints will help you in your C++/CX development. I recommend reviewing the Channel 9 videos on the topic, such as "Under the Covers with C++ for Metro-Style Apps" (bit.ly/1k7CWLq).

Libraries

The Standard Template Library (STL) is the most important library for C++ users. Templates and generics both answer the question, "How do you build type-safe generic containers?" It's templates in C++ and generics in C#, and though the syntax is somewhat similar, they're pretty different concepts. Templates are specialized at compile time. Generics are specialized at run time. As for C++/CLI, they're different enough in that they have different keywords and syntaxes. From a C# perspective, if you approach templates like you do generics you'll find they're somewhat familiar.

As the MSDN Library documentation (bit.ly/1bZzkTB) indicates, the STL establishes uniform standards for the application of iterators to STL containers or other sequences you define, by STL algorithms or other functions you define. This is a common definition of the STL, which means, of course, you'll need to understand what algorithms, iterators and containers are. I won't delve into these here, but they're well documented in the MSDN Library.

Keep in mind the C++ STL isn't the same thing as the C++ Standard Library. According to MSDN Library documentation, the Standard C++ Library in Visual Studio 2013 is a conforming implementation from which a C++ program can call on a large number of functions. These functions perform essential services such as input and output and provide efficient implementations of frequently used operations (bit.ly/1eQkPIS).



```
void DataBindingPresenter::Initialize(Object^ view)
{
    _view = safe_cast<DataBindingView^>(view);

    _logger->Log("Instantiate view model");
    _viewModel = ref new MainViewModel();

    _logger->Log("Attach to the View");
    _view->DataContext = _viewModel;

    _logger->Log("Initialize view model");
    _viewModel->Initialize();

    _logger->Log("Setting list values on view model");

    // In the real world a business logic layer would be called to
    // retrieve data. The BLL would access the DAL and return list
    auto dal = ref new MockDataDal();
    for (auto item : dal->GetCowboyList())
        _viewModel->NameValueList->Append(item);

    // Default to first record
    _viewModel->SelectedNameValue = _viewModel->NameValueList->GetAt(0);
}

void DataBindingPresenter::Navigate(Frame^ frame)
{
    _logger->Log("Navigating to view", "NAVIGATION", "5");
    frame->Navigate(TypeName(WsaXamlCpp::DataBindingView::typeid), this);
}

void DataBindingPresenter::CommandHandler(Object^ command)
{
    _logger->Log("CommandHandler executed", "COMMAND", "1");
    // We'll keep default behavior...
}

DataBindingPresenter::DataBindingPresenter()
{
    _logger = ref new DebugLogger();
}
```

```
public void Initialize(object view)
{
    _view = view as DataBindingView;

    _logger.Log("Instantiate view model");
    _viewModel = new MainViewModel();

    _logger.Log("Attach to the view model to the view");
    _view.DataContext = _viewModel;

    _logger.Log("Initializing view model");
    _viewModel.Initialize();

    _logger.Log("Setting list values on view model");

    // In the real world a business logic layer would be called to
    // retrieve data. The BLL would access the DAL and return list
    var dal = new MockDataDal();
    foreach (var item in dal.GetCowboyList())
        _viewModel.NameValueList.Add(item);

    // Default to last record
    _viewModel.SelectedNameValue = _viewModel.NameValueList.First();
}

public void Navigate(Frame frame)
{
    _logger.Log("Navigating to view", "NAVIGATION", "5");
    frame.Navigate(typeof(DataBindingView), this);
}

public void CommandHandler(object parameter)
{
    _logger.Log("CommandHandler executed", "COMMAND", "1");
    // We'll keep default behavior
}

public DataBindingPresenter()
{
    _logger = new DebugLogger();
}
```

```
'WsaXamlCpp.exe' (Win32): Loaded 'C:\Windows\assembly\NativeImages_v4.0.30319_32\System.Xml\e6b0fac6b8c9f63921dc57ccb85a0ee4\System.Xml.ni.dll'.
11/24/2013 03:25:54: NAVIGATION(5) - Navigating to view
11/24/2013 03:25:54: MISC(0) - Instantiate view model
'WsaXamlCpp.exe' (Win32): Loaded 'D:\DEV\Wrl\MSDN - WRL Article\MsdnExpress\Debug\WsaXamlCpp\AppX\WsaWrcCpp.dll'. Symbols loaded.
11/24/2013 03:25:55: MISC(0) - Attach to the View
11/24/2013 03:25:55: MISC(0) - Initialize view model
11/24/2013 03:25:55: MISC(0) - Setting list values on view model
```

Figure 2 **C++/CX and C# Code Using the Same View Model, Data Access Layer and Logger**

The Active Template Library (ATL) is a set of template-based C++ classes that lets you create small, fast COM objects. It has special support for key COM features, including stock implementations, dual interfaces, standard COM enumerator interfaces, connection points, tear-off interfaces and ActiveX controls (bit.ly/1engnjy). It's important to know that only a small subset of COM and the WINAPI are supported in a WinRT application. The WRL is better suited for WinRT applications.

The WRL is like a simplified ATL for the Windows Runtime. It consists of a set of headers and templates that can help you develop WinRT classes using standard C++ capabilities. It eliminates

Figure 3 **Contoso.idl**

```
import "inspectable.idl";
import "Windows.Foundation.idl";
import "ocidl.idl";
#define COMPONENT_VERSION 1.0

namespace Contoso {
  interface ICalculator;
  runtimeclass Calculator;

  [uuid(0be9429f-2c7a-40e8-bb0a-85bcb1749367), version(COMPONENT_VERSION),
    exclusiveto(Calculator)]
  interface ICalculator : IInspectable
  {
    // msdn.microsoft.com/library/jj155856
    // Walkthrough: Creating a Basic Windows Runtime Component Using WRL
    HRESULT Add([in] int a, [in] int b, [out, retval] int* value);
    HRESULT AddStr([in] HSTRING a, [in] HSTRING b, [out, retval] int* value);
  }
  [version(COMPONENT_VERSION), activatable(COMPONENT_VERSION)]
  runtimeclass Calculator
  {
    [default] interface ICalculator;
  }
}
```

a massive amount of boilerplate code you'd otherwise need to add and reduces the line count required to roughly the same as C++/CX. C++/CX is the way most developers are expected to produce and consume WinRT components, unless you need to do so in a code base that bans exceptions or has other special needs, or you simply prefer to avoid proprietary language extensions unless absolutely necessary. WRL is the lowest level of C++ development for interfacing to Windows Store applications. It does lack some facilities for building XAML components, so you should only use C++/CX or C# to create XAML components.

C++

Modern C++ (version 11 or higher) is significantly different from previous versions, outdating many current books, blogs and articles, so be careful. If something is using "new" or "delete," it's the old C++. Not that that's obsolete or useless, and you won't have to rewrite existing code, as C++11 plays well with existing code; you just won't be leveraging the simplicity and efficiency of modern C++.

C++/CX is a relatively C# developer-friendly way to author WinRT types. This language extension was created to make it easy to produce and consume WinRT types, but it isn't required. In fact, you can use WRL to do the same, but not as easily or directly. Note that C++/CX borrows the C++/CLI syntax, but targets a different runtime.

C++/CLI generally isn't used for Windows desktop development. Typically, it's used only for managed code in extremely narrow circumstances. For example, some existing native code performs a task well and you want to expose it for easy use from managed code, but the interface you want to expose to managed code isn't ideally exposed via COM or P/Invoke. The audience for C++/CLI is very, very small. I've noted it here only because of the similarities of the ^ (hat) notation and the potential for it to cause confusion as some CLI code samples won't be compatible with C++/CX.

Both C++/CLI and Windows Store applications written with C++/CX use the hat notation. You can use WinRT components in Windows desktop (not Windows Store) applications with C++/CX, in which case you use the hat. You can also use WinRT components in both Windows Store and Windows desktop applications using C++, in which case the applications won't use the hat.

When researching the Internet, it's a good idea to precede your searches with "modern C++" or "C++/CX" to ensure you'll be viewing applicable documentation.

A History Lesson

In the past, software reusability meant providing access to components without necessarily giving others access to the code. This could be done by providing a static linked library (.lib) file with a header (.h) file, which other developers could then use to link the .lib into different applications—the header provided the interface and the library the compiled sourcecode.

One of the inherent problems of this practice is the amount of memory and disk space it can take. For example, a 2MB library compiled into a software suite of six applications takes 12MB of disk space. Of course, the average XT-compatible PC disk capacity at the time was 20MB. With the emergence of the DLL came the ability to share a single set of code among multiple applications. With this, the six applications now take only 2MB to use the same library.

The details, pros and cons of static libraries versus DLLs are outside the scope of this article. You'll find exhaustive references to both libraries in MSDN Library documentation.

Static libraries and DLLs have a place in building modern C++ libraries. However, if you're trying to build reusable components that can be easily accessed by JavaScript, C++ and C# Windows Store applications, you need to resist the knee-jerk reaction to select the Windows Store application DLL (to which C# developers are accustomed). As you'll see later, it's not the DLL as you know it. Instead, you'll want to select the project template for the WinRT component (which I refer to as WRC). This is what Microsoft recommends, and it provides a development experience closest to the one to which you're accustomed.

Project Templates

For C# developers, adding a new library is simple—you add a class library project and use it to build your interfaces, classes and so forth. Referencing class libraries is just as easy, as you simply add references to your class library project from your application or other class libraries—productivity at its best.

Unfortunately, building C++ libraries is far less straightforward and, as of this writing, you won't find a lot of documentation or examples on how to set up Visual Studio for the development of reusable components, in particular on configuring the various available project template types. The following will help you set up Visual Studio to use the Windows Store application C++ project templates.

WinRT Component

The MSDN Library documentation for "DLLs (C++/CX)" suggests that when you need to create a DLL for a Windows Store app, you should create it as a WinRT component by using the project template of that name (bit.ly/1iwL1Wg). As I noted, this option offers the most familiar experience for adding and referencing libraries.

As **Figure 2** shows, there are a lot of similarities between C++/CX and C#, so C# developers should be able to quickly ramp up

/n software

We didn't invent the Internet...

...but our components help you power the apps that bring it to business.

TOOLS • COMPONENTS • ENTERPRISE ADAPTERS

- E-Business
  AS2, EDI/X12, NAESB, OFTP ...
- Credit Card Processing
  Authorize.Net, TSYS, FDMS ...
- Shipping & Tracking
  FedEx, UPS, USPS ...
- Accounting & Banking
  QuickBooks, OFX ...
- Internet Business
  Amazon, eBay, PayPal ...
- Internet Protocols
  FTP, SMTP, IMAP, POP, WebDav ...
- Secure Connectivity
  SSH, SFTP, SSL , Certificates ...
- Secure Email
  S/MIME, OpenPGP ...
- Network Management
  SNMP, MIB, LDAP, Monitoring ...
- Compression & Encryption
  Zip, Gzip, Jar, AES ...

Microsoft .NET | Microsoft BizTalk Server | Microsoft SQL Server | Windows Mobile | activex | Microsoft SharePoint 2010 | CodeGear Delphi | CodeGear C++Builder | Java | php

The Market Leader in Internet Communications, Security, & E-Business Components

Each day, as you click around the Web or use any connected application, chances are that directly or indirectly some bits are flowing through applications that use our components, on a server, on a device, or right on your desktop. It's your code and our code working together to move data, information, and business. We give you the most robust suite of components for adding Internet Communications, Security, and E-Business Connectivity to any application, on any platform, anywhere, and you do the rest. Since 1994, we have had one goal: to provide the very best connectivity solutions for our professional developer customers. With more than 100,000 developers worldwide using our software and millions of installations in almost every Fortune 500 and Global 2000 company, our business is to connect business, one application at a time.

connectivity
powered by

To learn more please visit our website → www.nsoftware.com

Microsoft .NET, | Java | ActiveX | C++ | Delphi | C++ Builder | PHP | Objective-C | Windows | Windows Mobile | Mac OS X | Linux/Unix | iPhone to C++. The interesting thing about the WinRT component is that it's transparent to C# and C++ Windows Store apps, giving C# developers a means to focus on both productivity and performance as the situation dictates. Thus, as I indicated earlier, if you want to reuse an existing C# Logger component in your C++ XAML Windows Store application, all you have to do is port the code into a C# WRC project and you're done. You can now reference it from both C++ and C# applications (as I've done in the sample solution).

Keep in mind that because the ABI does impose some restrictions, such as requiring classes to be sealed, you'll have to look at development from a new angle, one that emphasizes "composition over abstraction," which lends itself nicely to building composite applications.

WRL Class Library Project Template

A WRL Class Library project template is a set of user-defined classes built using the WRL. Unfortunately, WRL tutorials and explanations can be quite daunting. As an introduction to WRL and understanding the WRL Class Library project template, I recommend you follow the tutorial "Walkthrough: Creating a Basic Windows Runtime Component Using WRL" (bit.ly/1n1gW2). You'll be impressed at how, with just a few lines of code, you can access your C++ code from a JavaScript application by simply adding a reference to the library. This tutorial was the basis for the sample solution I've provided.

Figure 4 **Contoso.cpp**

```
#include "pch.h"
#include "Contoso_h.h"
#include <wrl.h>
#include <string>
#include <memory>
#include "Calculator.h"
using namespace std;
using namespace Microsoft::WRL;
using namespace Windows::Foundation;

namespace ABI {
  namespace Contoso  {
    class Calculator : public RuntimeClass<ICalculator> {
      InspectableClass(RuntimeClass_Contoso_Calculator, BaseTrust)

    public:
      // Use an external C++ Windows Store App DLL to handle strings
      // note: WRL doesn't permit overloading functions
      HRESULT __stdcall AddStr(_In_ HSTRING a, _In_ HSTRING b, _Out_ int* value)
      {
        // Convert HSTRING values into const wchar_t*
        // so you can pass them into C++ DLL
        const wchar_t* buffera = WindowsGetStringRawBuffer(a, nullptr);
        const wchar_t* bufferb = WindowsGetStringRawBuffer(b, nullptr);
        // Instantiate calculator using modern methods - reference
        // msdn.microsoft.com/library/hh279669
        auto calc = make_shared<WsaDllCpp::Calculator>();
        // Add the string values
        auto val = calc->Add(buffera, bufferb);
        // Assign value
        *value = val;
        return S_OK;
      }
      // msdn.microsoft.com/library/jj155856
      // Walkthrough: Creating a Basic Windows Runtime Component Using WRL
      HRESULT __stdcall Add(_In_ int a, _In_ int b, _Out_ int* value)   {
        if (value == nullptr)
        {
          return E_POINTER;
        }
        *value = a + b;
        return S_OK;
            }
      };
      ActivatableClass(Calculator);
  }
}
```

The tutorial is important for those new to WRL because it also provides an important link for generating a WRL project using the WRL Class Library template. This template includes the necessary information to invoke the MIDL compiler, which processes an IDL file (see bit.ly/1fLMurc for more information). Behind the scenes, the MIDL compiler generates the necessary (hidden) files for the Contoso WRL project, which take care of all the overhead associated with WRL development—contoso_h.h, contoso_i.c and contoso_p.c. As you'll see from the tutorial, all you have to create are the contoso.idl and contoso.cpp files, which **Figure 3** and **Figure 4** show, respectively. Note that I added the AddStr(string, string) functionality; it wasn't part of the tutorial.

As with C# class libraries, using a WRL component from an external project or application simply requires you to add a reference to this project.

Static Libraries

Static libraries are appropriate when you're rebuilding code using current tools for each project and you want minimal overhead calling into the code. This is the option to choose if you probably won't be using the exact same binary across multiple projects built at different times.

The compiler needs to know the location of the static library (.lib) and its associated header (.h) in order to successfully link the compiled code into the application. This can be done by right-clicking on the Solution and selecting Common Properties | Project Dependencies and then your project (WsaWrcCpp) from the projects list, and checking the .lib file (WsaLibCpp) as a dependency. Then, right-click on the Project (WsaWrcCpp), select Properties | Common Properties | References | Add New Reference | Solution| Projects, and then choose your .lib file (WsaLibCpp).

You now have to let the project know where to find the .lib projects header file. If you're running through the sample solution, right-click on the WsaWrcCpp (WRC) project and select Properties | Configuration Properties | C/C++ | General, and set the "Additional Include Directories" to the location of the header file, in this case, that would be $(SolutionDir)WsaLibCpp;<existing paths>.

DLLs

In the downloadable sample solution you'll find a Windows Store application DLL for a C++ project named WsaDllCpp (as in **Figure 2**). I used the MSDN Library article, "Walkthrough: Creating and Using a Dynamic Link Library (C++)" (bit.ly/1enxzWc) as a reference. Note that this project is for demonstration purposes only as the use of this DLL has been discouraged (in my correspondence with Microsoft C++ developers and reviewers). I've included it because you may need to use existing DLLs in your Windows Store application.

With that in mind, the WsaDllCpp DLL gives the solution Contoso WRL (the WRL Class Library template discussed earlier) with the ability to add two strings together; for example, WsaDllCpp::Calculator::Add("1000","2000") yields 3000. The following interface code highlights the difference between a C++ DLL over the C# DLL as you know it:

```
#pragma once

#include <string>
#define WSADLLCPP_API __declspec(dllexport)

namespace WsaDllCpp
{
  class Calculator
  {
  public:
    WSADLLCPP_API int Add(const wchar_t* numberOne, const wchar_t* numberTwo);
    WSADLLCPP_API int Add(int numberOne, int numberTwo);
  };
}
```

As you can see, there's a wee bit of overhead associated with making functions visible to external libraries and applications (__declspec). Likewise, the settings required to successfully compile the WsaDllCpp into the application go beyond simply setting a project reference. I had to right-click on the Contoso WRL project and select Properties | Configuration Properties | C/C++ | General, and set the "Additional Include Directories" to the location of the header file.

C# developers might find adding a reference to a DLL isn't intuitive, or at least it isn't what you're accustomed to, as there's usually a "reference" link in the project list on which to right-click (which you'll find in the WRC project). As with the .lib file, to add a reference you'll have to right-click on the project and then select Add | Reference, and then click the Add new Reference button.

Unlike with C# development, you can't assume the paths you select will be converted to relative paths—they won't. It's best to get used to using the Visual Studio macros to specify your paths. For the previous example I had to set the "Additional Include Directories" to $(SolutionDir)WsaDllCpp; <existing paths>.

Wrapping Up

Once you hone in on that correct bookshelf, you'll discover a wealth of information (particularly in the MSDN Library), making your learning process more efficient and rewarding. This first step toward a basic understanding of the C++ domain will help you in your research and development of C++ Windows Store applications. There's just so much information that doesn't apply (such as legacy C++, MFC, C++/CLI and, for the most part, ATL) that this awareness will prevent you from wasting valuable cycles on material that could take you down a long road to a dead end.

I also recommend you review Kenny Kerr's *MSDN Magazine* articles (bit.ly/1iv7mUQ) and Channel 9 (bit.ly/1dFqYjV), as well as Michael B. Mclaughlin's articles at bit.ly/1b5CDhs. (A valuable resource not listed on this site is his "C# to C++ - A Somewhat Short Guide," available at bit.ly/MvdZv4.)

If you focus your attention on C++/CX, modern C++ and on using the WinRT component, you'll be well on your way to achieving your goals in the Windows Store application development environment. ■

Bill Kratochvil *is owner of Global Webnet LLC, an Amarillo, Texas-based company that performs contract software architecture and development. Currently, Kratochvil is on contract with a leading company in the medical industry.*

THANKS to the following Microsoft technical experts for reviewing this article: *Mohammad Al-Sabt, John Cuyle, Chris Guzak, Frantisek Kaduk, Ben Kuhn and Thomas Petchel. Also, thanks to Steven Southwick.*

Multithreading and Dispatching in MVVM Applications

Laurent Bugnion

Just about a year ago I began a series of articles on the Model-View-ViewModel (MVVM) pattern for the *MSDN Magazine* Web site (you can access them all at is.gd/mvvmmsdn). The articles show how to use the components of the MVVM Light Toolkit to build loosely coupled applications according to this pattern. I explore the dependency injection (DI) and inversion of control (IOC) container patterns (including the MVVM Light SimpleIoc), introduce the Messenger and discuss View services (such as Navigation, Dialog and so forth). I also show how to create design time data to maximize the use of visual designers such as Blend, and I talk about the RelayCommand and EventToCommand components that replace event handlers for a more decoupled relationship between the View and its ViewModel.

In this article, I want to delve into another frequent scenario in modern client applications—handling multiple threads and helping them communicate with each other. Multithreading is an increasingly important topic in modern application frameworks such as Windows 8, Windows Phone, Windows Presentation Foundation (WPF), Silverlight and more. On every one of these platforms, even the least powerful, it's necessary to start background threads and to manage them. In fact, you could argue that it's even more important on small platforms with less computing power, in order to offer an enhanced UX.

The Windows Phone platform is a good example. In the very first version (Windows Phone 7), it was quite difficult to get smooth scrolling in long lists, especially when the item templates contained images. However, in later versions the decoding of images, as well as some animations, are passed to a dedicated background thread. As a result, when an image is loaded, it doesn't impact the main thread anymore, and the scrolling remains smooth.

This example underlines some important concepts I'll explore in this article. I'll start by reviewing how multithreading works in XAML-based applications in general.

Simply put, a thread can be considered as a smaller unit of execution of an application. Each application owns at least one thread, which is called the main thread. This is the thread that gets started by the OS when the main method of the application is called, on startup. Note that more or less the same scenario happens on all supported platforms, as much on WPF running on powerful computers as on Windows Phone-based devices with limited computing power.

When a method is called, the operation is added to a queue. Each operation is executed sequentially, according to the order in which it was added to the queue (though it's possible to influence the order in which the operations are executed by assigning a priority to them.). The object responsible for managing the queue is called the thread's dispatcher. This object is an instance of the Dispatcher class in WPF, Silverlight and Windows Phone. In Windows 8, the dispatcher object is named CoreDispatcher and uses a slightly different API.

This article discusses:

- How multithreading works in XAML-based applications
- Allowing threads to communicate with each other
- Dispatching in MVVM applications
- A real-life dispatching example using the Windows Phone Compass Sensor

Technologies discussed:

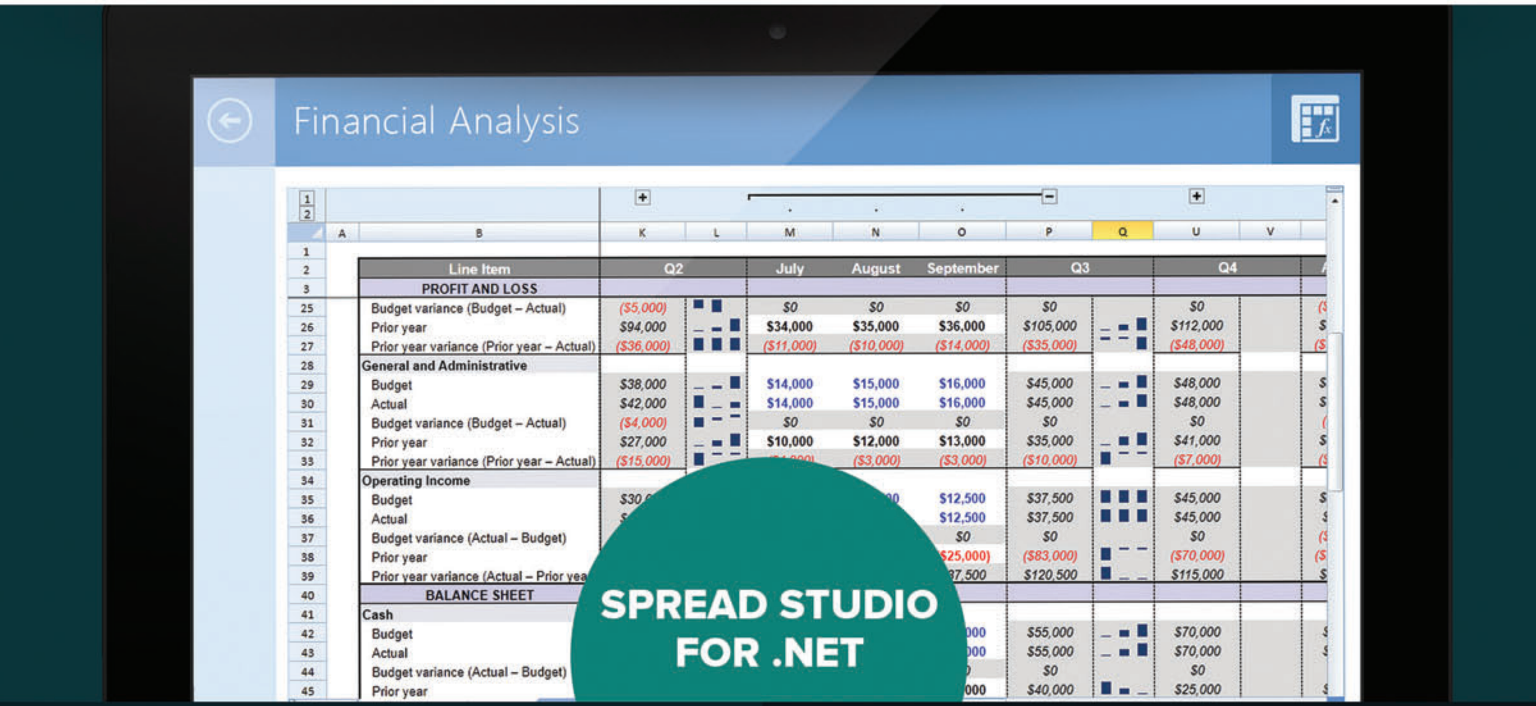
Windows 8, MVVM Light Toolkit, Visual Studio, Windows Phone, Windows Presentation Foundation

Code download available at:

msdn.microsoft.com/magazine/msdnmag0414

THE EXPERTS IN SPREADSHEETS

When you need the power of a spreadsheet combined with the functionally of an advanced data grid, you need Spread, the world's #1 selling spreadsheet component for Microsoft Visual Studio development.



Financial Analysis

	Line Item	Q2	July	August	September	Q3	Q4
3	PROFIT AND LOSS						
25	Budget variance (Budget – Actual)	($5,000)	$0	$0	$0	$0	$0
26	Prior year	$94,000	$34,000	$35,000	$36,000	$105,000	$112,000
27	Prior year variance (Prior year – Actual)	($36,000)	($11,000)	($10,000)	($14,000)	($35,000)	($48,000)
28	General and Administrative						
29	Budget	$38,000	$14,000	$15,000	$16,000	$45,000	$48,000
30	Actual	$42,000	$14,000	$15,000	$16,000	$45,000	$48,000
31	Budget variance (Budget – Actual)	($4,000)	$0	$0	$0	$0	$0
32	Prior year	$27,000	$10,000	$12,000	$13,000	$35,000	$41,000
33	Prior year variance (Prior year – Actual)	($15,000)		($3,000)	($3,000)	($10,000)	($7,000)
34	Operating Income						
35	Budget				$12,500	$37,500	$45,000
36	Actual				$12,500	$37,500	$45,000
37	Budget variance (Actual – Budget)				$0	$0	$0
38	Prior year					($83,000)	($70,000)
39	Prior year variance (Actual – Prior yea					$120,500	$115,000
40	BALANCE SHEET						
41	Cash						
42	Budget					$55,000	$70,000
43	Actual					$55,000	$70,000
44	Budget variance (Actual – Budget)					$0	$0
45	Prior year					$40,000	$25,000

SPREAD STUDIO FOR .NET

Spread provides a flexible and familiar spreadsheet/grid architecture, advanced charting, and a powerful formula library that is ideal for creating financial modeling and risk analysis, budgeting, insurance, scientific, and many other applications.

AVAILABLE FOR:

Windows Forms · ASP.NET · WPF · Silverlight · WinRT

- Read Microsoft Excel files and/or generate Excel output from your business application
- Provide Excel like characteristics, such as complete formula calculation support, enhanced filtering, sorting, conditional formatting, cell types, sheets, and more
- Create Complex Form/Data-entry Layouts that include many fields and calculations, such as insurance forms or tax forms
- Add Data Visualization, Analysis, and Dashboards

ComponentOne®
a division of GrapeCity®

DOWNLOAD YOUR FREE TRIAL

▸ componentone.com

Figure 1 **Asynchronous Operation in the Microsoft .NET Framework**

```
public void DoSomethingAsynchronous()
{
  var loopIndex = 0;

  ThreadPool.QueueUserWorkItem(
    o =>
    {
      // This is a background operation!
      while (_condition)
      {
        // Do something
        // ...

        // Sleep for a while
        Thread.Sleep(500);
      }
  });
}
```

As required by the application, new threads can be started explicitly in code, implicitly by some libraries or by the OS. Mostly, the purpose of starting a new thread is to execute an operation (or wait on the result of an operation) without blocking the rest of the application. This can be the case of a computationally intensive operation, an I/O operation and so on. This is why modern applications are increasingly multithreaded, because UX requirements are also increasing. As applications become more complex, the number of threads they start increases. A good example of this trend is the Windows Runtime framework used in Windows Store apps. In these modern client applications, asynchronous operations (operations running on background threads) are very common. For instance, every file access in Windows 8 is now an asynchronous operation. Here's how a file gets read (synchronously) in WPF:

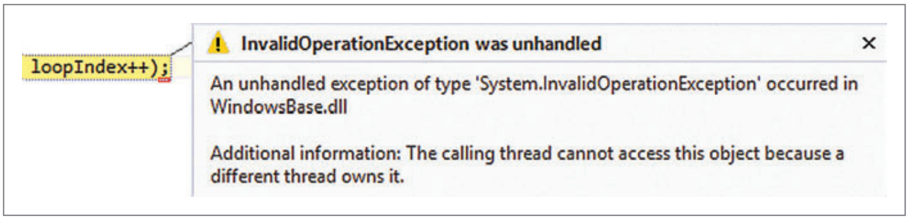
```
public string ReadFile(FileInfo file)
{
  using (var reader = new StreamReader(file.FullName))
  {
    return reader.ReadToEnd();
  }
}
```

And here's the equivalent (asynchronous) operation in Windows 8:

```
public async Task<string> ReadFile(IStorageFile file)
{
  var content = await FileIO.ReadTextAsync(file);
  return content;
}
```

Note the presence of the await and async keywords in the Windows 8 version. They're there to avoid using callbacks in asynchronous operations, and to make the code easier to read. They're needed here because the file operation is asynchronous. The WPF version, in contrast, is synchronous, which risks blocking the main thread if the file that's getting read is long. This can cause choppy animations, or a lack of update on the UI, which worsens the UX.

Similarly, long operations in your applications should be taken care of on a background thread if they risk making the UI choppy. For example, in WPF, Silverlight and Windows Phone, the code in **Figure 1** initiates a background operation that runs a long loop. In every loop, the thread is put to sleep for a short time to give time to the other threads to process their own operations.



```
loopIndex++);
```

⚠ **InvalidOperationException was unhandled** ×

An unhandled exception of type 'System.InvalidOperationException' occurred in WindowsBase.dll

Additional information: The calling thread cannot access this object because a different thread owns it.

Figure 2 **Cross-Thread Exception Dialog**

Letting Threads Communicate

When a thread needs to communicate with another thread, some precautions need to be taken. For example, I'll modify the code in **Figure 1** to display a status message to the user on each loop. To do this, I just add a line of code in the while loop, which sets the Text property of a StatusTextBlock control located in the XAML:

```
while (_condition)
{
  // Do something

  // Notify user
  StatusTextBlock.Text = string.Format("Loop # {0}", loopIndex++);

  // Sleep for a while
  Thread.Sleep(500);
}
```

The application named SimpleMultiThreading that accompanies this article shows this example. If you run the application using the button labeled "Start (crashes the app)," the application, indeed, crashes. So what happened? When an object is created, it belongs to the thread on which the constructor method was called. For UI elements, objects are created by the XAML parser when the XAML document is loaded. This all happens on the main thread. As a consequence, all the UI elements belong to the main thread, which is also often called the UI thread. When the background thread in the previous code attempts to modify the Text property of the StatusTextBlock, this creates an illegal cross-thread access. As a consequence, an exception is thrown. This can be shown by running the code in a debugger. **Figure 2** shows the exception dialog. Notice the "Additional information" message, which indicates the root of the problem.

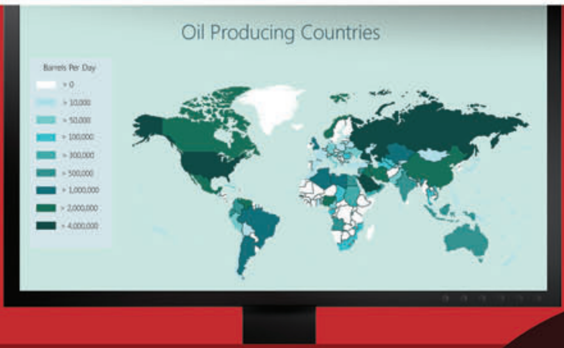
In order for this code to work, the background thread needs to queue the operation on the main thread by contacting its dispatcher. Thankfully, each FrameworkElement is also a DispatcherObject as shown by the .NET class hierarchy in **Figure 3**. Every DispatcherObject exposes a Dispatcher property that gives access to its owner dispatcher. Thus, the code can be modified as shown in **Figure 4**.

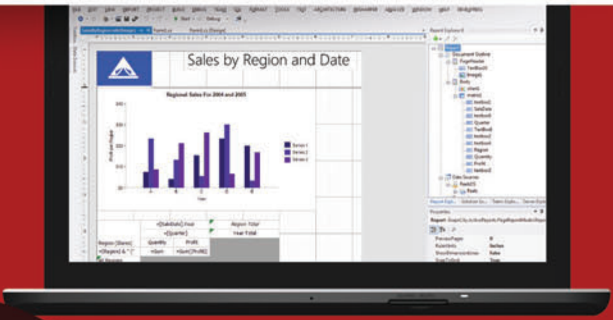
Dispatching in MVVM Applications

When the background operation is executed from a ViewModel, things are a little different. Typically, ViewModels don't inherit from DispatcherObject. They're plain old CLR objects (POCOs) that implement the INotifyPropertyChanged interface. For example, **Figure 5** shows a ViewModel deriving from the MVVM Light ViewModelBase class. In true MVVM manner, I add an observable property named Status that raises the PropertyChanged event. Then, from the background thread code, I attempt to set this property with an information message.

REPORTS ANYWHERE

We are the Reporting experts and we care about performance and Visual Studio integration as much as you. The all new ActiveReports 8 reporting engine delivers a granular API that allows you to create concise and beautiful reports. Whether you need multiple outputs, the ability to create maps, barcodes, or the power of HTML5 to display your reports, ActiveReports 8 can handle all of your heavy reporting requirements.





ALL NEW ACTIVEREPORTS 8

Sophisticated, Fast & Powerful Reports

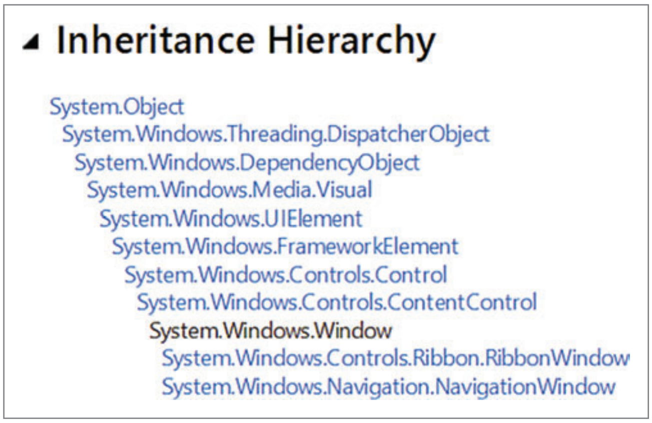
Flexible Layouts Using Section, Region & Fixed Page Designers

HTML5 & Touch Optimized Report Viewers for Mobile Devices

A New Scalable, Distributed & Load Balanced Enterprise-grade Report Server

ComponentOne®
a division of GrapeCity®

DOWNLOAD YOUR FREE TRIAL
▶ componentone.com



Inheritance Hierarchy

System.Object
System.Windows.Threading.DispatcherObject
System.Windows.DependencyObject
System.Windows.Media.Visual
System.Windows.UIElement
System.Windows.FrameworkElement
System.Windows.Controls.Control
System.Windows.Controls.ContentControl
System.Windows.Window
System.Windows.Controls.Ribbon.RibbonWindow
System.Windows.Navigation.NavigationWindow

Figure 3 **Window Class Hierarchy**

Running this code in Windows Phone or Silverlight works fine until I try to data bind the Status property to a TextBlock in the XAML front end. Running the operation again crashes the application. Just like before, as soon as the background thread attempts to access an element belonging to another thread, the exception is thrown. This occurs even if the access is done through data binding.

Note that in WPF, things are different and the code shown in **Figure 5** works even if the Status property is data-bound to a TextBlock. This is because WPF automatically dispatches the PropertyChanged event to the main thread, unlike all the other XAML frameworks. In all other frameworks, a dispatching solution is needed. In fact, what's really needed is a system that dispatches the call only if necessary. In order to share the ViewModel code between WPF and other frameworks, it would be great if you didn't have to care about the need to dispatch, but had an object that would do this automatically.

Because the ViewModel is a POCO, it doesn't have access to a Dispatcher property, so I need another way to access the main thread and to enqueue an operation. This is the purpose of the MVVM Light DispatcherHelper component. In essence, what this class does is store the main thread's Dispatcher in a static property and expose a few utility methods to access it in a convenient and consistent manner. In order to be functional, the class needs to be initialized on the main thread. Ideally, this should be done early in the application's lifetime, so the features are accessible from the application's start. Typically, in an MVVM Light application, the DispatcherHelper is initialized in App.xaml.cs, which is the file defining the startup class of the application. In Windows Phone, you call DispatcherHelper.Initialize in the InitializePhoneApplication method, right after the application's main frame is created. In WPF, the class is initialized in the App constructor. In Windows 8, you call the Initialize method in OnLaunched, right after the windows is activated.

After the call to the DispatcherHelper.Initialize method is complete, the UIDispatcher property of the DispatcherHelper class contains a reference to the main thread's dispatcher. It's relatively rare to use the property directly, but it is possible if needed. Instead, however, it's better to use the CheckBeginInvokeOnUi method. This method takes a delegate as parameter. Typically, you use a lambda expression as shown in **Figure 6**, but it could also be a named method.

As the name suggests, this method performs a check first. If the caller of the method is already running on the main thread, no dispatching is necessary. In that case, the delegate is executed immediately, directly on the main thread. If, however, the caller is on a background thread, the dispatching is executed.

Because the method checks before dispatching, the caller can rely on the fact the code will always use the optimal call. This is especially useful when you're writing cross-platform code, where multithreading might work with small differences on different platforms. In that case, the ViewModel code shown in **Figure 6** can be shared anyway, without any need to modify the line where the Status property is set.

Figure 4 **Dispatching the Call to the UI Thread**

```
while (_condition)
{
  // Do something

  Dispatcher.BeginInvoke(
    (Action)(() =>
    {
      // Notify user
      StatusTextBlock.Text = string.Format("Loop # {0}", loopIndex++);
    }));

  // Sleep for a while
  Thread.Sleep(500);
}
```

Figure 5 **Updating a Bound Property in the ViewModel**

```
public class MainViewModel : ViewModelBase
{
  public const string StatusPropertyName = "Status";

  private bool _condition = true;
  private RelayCommand _startSuccessCommand;
  private string _status;

  public RelayCommand StartSuccessCommand
  {
    get
    {
      return _startSuccessCommand
        ?? (_startSuccessCommand = new RelayCommand(
          () =>
          {
            var loopIndex = 0;
            ThreadPool.QueueUserWorkItem(
              o =>
              {
                // This is a background operation!

                while (_condition)
                {
                  // Do something

                  DispatcherHelper.CheckBeginInvokeOnUI(
                    () =>
                    {
                      // Dispatch back to the main thread
                      Status = string.Format("Loop # {0}", loopIndex++);
                    });

                  // Sleep for a while
                  Thread.Sleep(500);
                }
              });
          }));
    }
  }

  public string Status
  {
    get
    {
      return _status;
    }
    set
    {
      Set(StatusPropertyName, ref _status, value);
    }
  }
}
```

In addition, DispatcherHelper abstracts the differences in the dispatcher API between the XAML platforms. In Windows 8, the CoreDispatcher's main members are the RunAsync method and the HasThreadAccess property. In other XAML frameworks, however, the BeginInvoke and CheckAccess methods are used, respectively. By using the DispatcherHelper, you don't have to worry about these differences, and can more easily share the code.

Real-Life Dispatching: Sensors

I'll illustrate the use of DispatcherHelper by building a Compass sensor Windows Phone application.

The sample code accompanying this article contains a draft application named CompassSample - Start. When you open this application in Visual Studio, the access from the MainViewModel to the compass sensor is encapsulated in a service called SensorService, which is an implementation of the ISensorService interface. These two elements can be found in the Model folder.

The MainViewModel gets a reference to the ISensorService in its constructor and registers for every compass change using the SensorService RegisterForHeading method. This method requires a callback, which will be executed every time the sensor reports a change in the Windows Phone-based device heading. In the MainViewModel, replace the default constructor with the following code:

```
sensorService.RegisterForHeading(
  heading =>
  {
    Heading = string.Format("{0:N1}°", heading);
    Debug.WriteLine(Heading);
  });
```

Unfortunately, there's no way to simulate the device compass in the Windows Phone emulator. To test the code, you'll need to run the app on a physical device. Connect a developer device and run the code in debug mode by clicking F5. Observe the Output console in Visual Studio. You'll see the output of the Compass listed. If you move the device, you'll be able to find the north and observe how the value keeps updating.

Next, I'll bind a TextBlock in XAML to the Heading property in the MainViewModel. Open the MainPage.xaml and locate the TextBlock located in the ContentPanel. Replace the "Nothing yet" in the Text property with "{Binding Heading}". If you run the application again, in debug mode, you'll witness a crash with an error message similar to the earlier one. Again, this is a cross-thread exception.

The error is thrown because the compass sensor runs on a background thread. When the callback code is called, it also runs on the background thread, as does the setter of the Heading property. Because the TextBlock belongs to the main thread, the exception is thrown. Here, too, you need to create a "safe zone" to take care of dispatching the operations to the main thread. To do this, open the SensorService class. The CurrentValueChanged event is handled by a method named CompassCurrentValueChanged; this is where the callback method is executed. Replace this code with the following, which uses DispatcherHelper:

```
void CompassCurrentValueChanged(
  object sender,
  SensorReadingEventArgs<CompassReading> e)
{
  if (_orientationCallback != null)
  {
    DispatcherHelper.CheckBeginInvokeOnUI(
      () => _orientationCallback(e.SensorReading.TrueHeading));
  }
}
```

Figure 6 **Using DispatcherHelper to Avoid Crashing**

```
while (_condition)
{
  // Do something

  DispatcherHelper.CheckBeginInvokeOnUI(
    () =>
    {
      // Dispatch back to the main thread
      Status = string.Format("Loop # {0}", loopIndex++);
    });

  // Sleep for a while
  Thread.Sleep(500);
}
```

Now the DispatcherHelper must be initialized. To do this, open App.xaml.cs and locate the method called InitializePhoneApplication. At the very end of this method, add DispatcherHelper.Initialize();. Running the code now produces the expected result, displaying the Windows Phone-based device heading properly.

Note that not all sensors in Windows Phone raise their events on a background thread. The GeoCoordinateWatcher sensor, for instance, which is used to observe the phone's geolocation, already returns on the main thread for your convenience. By using DispatcherHelper, you don't have to worry about this and can always call the main thread's callback in the same manner.

Wrapping Up

I discussed how the Microsoft .NET Framework handles threads and what precautions need to be taken when a background thread wants to modify an object created by the main thread (also called the UI thread). You saw how this can cause a crash, and that to avoid this crash, the main thread's Dispatcher should be used to properly handle the operation.

Then I translated this knowledge to an MVVM application and introduced the DispatcherHelper component of the MVVM Light Toolkit. I showed how you can use this component to avoid issues when communicating from a background thread, and how it optimizes this access and abstracts the differences between WPF and the other XAML-based frameworks. By doing so, it allows easy sharing of ViewModel code and makes your work easier.

Finally, I demonstrated in a real-life example how the DispatcherHelper can be used in a Windows Phone application to avoid issues when you're working with certain sensors that raise their events on a background thread.

In the next article, I'll delve further into the Messenger component of MVVM Light, and show how it can be used for easy communication among objects without any need for them to know about each other, in a true decoupled manner. ■

Laurent Bugnion *is senior director for IdentityMine Inc., a Microsoft partner working with technologies such as Windows Presentation Foundation, Silverlight, Pixelsense, Kinect, Windows 8, Windows Phone and UX. He's based in Zurich, Switzerland. He is also a Microsoft MVP and a Microsoft Regional Director.*

Thanks to the following technical expert for reviewing this article: *Thomas Petchel (Microsoft)*

Patterns for Asynchronous MVVM Applications: Commands

Stephen Cleary

This is the second article in a series on combining async and await with the established Model-View-ViewModel (MVVM) pattern. Last time, I showed how to data bind to an asynchronous operation, and I developed a key type called NotifyTaskCompletion<TResult> that acted like a data binding-friendly Task<TResult> (see msdn.microsoft.com/magazine/dn605875). Now I'll turn to ICommand, a .NET interface used by MVVM applications to define a user operation (which is often data bound to a button), and I'll consider the implications of making an asynchronous ICommand.

The patterns here may not fit every scenario perfectly, so feel free to tune them to your needs. In fact, this entire article is presented as a series of improvements on an asynchronous command type. At the end of these iterations, you'll end up with an application like what's shown in **Figure 1**. This is similar to the application developed in my last article, but this time I provide the user with an actual command to execute. When the user clicks the Go button, the URL is read from the textbox and the application will count the number of bytes at that URL (after an artificial delay). While the operation is in progress, the user may not start another one, but he can cancel the operation.

This article discusses:

- The ICommand interface
- Handling asynchronous command completion via data binding
- Adding the ability to cancel a command
- Creating a simple work queue

Technologies discussed:

Asynchronous programming, Model-View-ViewModel Pattern

Code download available at:

msdn.microsoft.com/magazine/msdnmag0414

> The patterns here may not fit every scenario perfectly, so feel free to tune them to your needs.

I'll then show how a very similar approach can be used to create any number of operations. **Figure 2** illustrates the application modified so the Go button represents adding an operation to a collection of operations.

There are a couple of simplifications I'm going to make during the development of this application, to keep the focus on asynchronous commands instead of implementation details. First, I won't use command execution parameters. I've hardly ever needed to use parameters in real-world apps; but if you need them, the patterns in this article can be easily extended to include them. Second, I don't implement ICommand.CanExecuteChanged myself. A standard field-like event will leak memory on some MVVM platforms (see bit.ly/1bROnVj). To keep the code simple, I use the Windows Presentation Foundation (WPF) built-in CommandManager to implement CanExecuteChanged.

I'm also using a simplified "service layer," which for now is just a single static method, as shown in **Figure 3**. It's essentially the same service as in my last article, but extended to support cancellation. The next article will deal with proper asynchronous service design, but for now this simplified service will do.

Asynchronous Commands

Before getting started, take a quick look at the ICommand interface:

```
public interface ICommand
{
  event EventHandler CanExecuteChanged;
  bool CanExecute(object parameter);
  void Execute(object parameter);
}
```

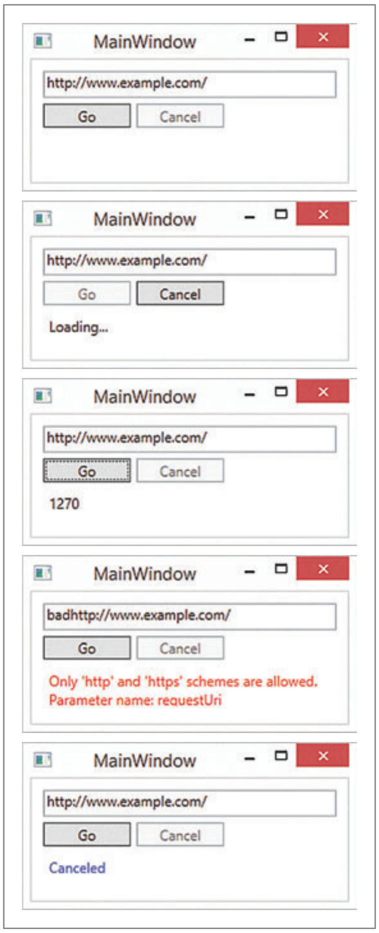
Ignore CanExecuteChanged and the parameters, and think for a bit about how an asynchronous command would work with this interface. The CanExecute method must be synchronous; the only member that can be asynchronous is Execute. The Execute method was designed for synchronous implementations, so it returns void. As I mentioned in a previous article, "Best Practices in Asynchronous Programming" (msdn.microsoft.com/magazine/jj991977), async void methods should be avoided unless they're event handlers (or the logical equivalent of event handlers). Implementations of ICommand.Execute are logically event handlers and, thus, may be async void.

However, it's best to minimize the code within an async void method and expose an async Task method instead that contains the actual logic. This practice makes the code more testable. With this in mind, I propose the following as an asynchronous command interface, and the code in **Figure 4** as the base class:

```
public interface IAsyncCommand : ICommand
{
  Task ExecuteAsync(object parameter);
}
```

The base class takes care of two things: It punts the CanExecuteChanged implementation off to the CommandManager class; and it implements the async void ICommand.Execute method by calling the IAsyncCommand.ExecuteAsync method. It awaits the result to ensure that any exceptions in the asynchronous command logic will be properly raised to the UI thread's main loop.

This is a fair amount of complexity, but each of these types has a purpose. IAsyncCommand can be used for any asynchronous ICommand implementation, and is intended to be exposed from ViewModels and consumed by the View and by unit tests. AsyncCommandBase handles some of the common boilerplate code common to all asynchronous ICommands.



http://www.example.com/

http://www.example.com/

Loading...

http://www.example.com/

1270

badhttp://www.example.com/

Only 'http' and 'https' schemes are allowed.
Parameter name: requestUri

http://www.example.com/

Canceled

Figure 1 **An Application That Can Execute One Command**

With this groundwork in place, I'm ready to start developing an effective asynchronous command. The standard delegate type for a synchronous operation without a return value is Action. The asynchronous equivalent is Func<Task>. **Figure 5** shows my first iteration of a delegate-based AsyncCommand.

At this point, the UI has only a textbox for the URL, a button to start the HTTP request and a label for the results. The XAML and the essential parts of the ViewModel are simple. Here's MainWindow.xaml (skipping the positioning attributes such as Margin):

```
<Grid>
  <TextBox Text="{Binding Url}" />
  <Button Command="{Binding CountUrlBytesCommand}" Content="Go" />
  <TextBlock Text="{Binding ByteCount}" />
</Grid>
```

MainWindowViewModel.cs is shown in **Figure 6**.

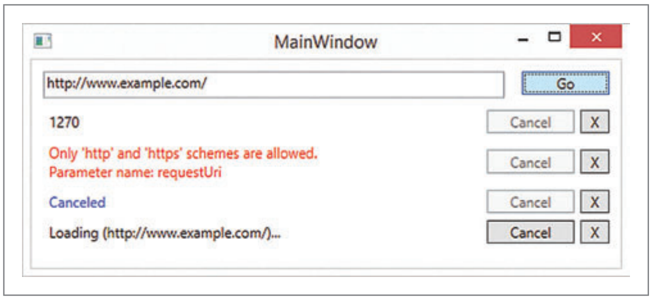
If you execute the application (AsyncCommands1 in the sample code download), you'll notice four cases of inelegant behavior. First, the label always shows a result, even before the button is clicked. Second, there's no busy indicator after you click the button to indicate the operation is in progress. Third, if the HTTP request faults, the exception is passed to the UI main loop, causing an application crash. Fourth, if the user makes several requests, she can't distinguish the results; it's possible for the results of an earlier request to overwrite the results of a later request due to varying server response times.

This is quite a slew of problems! But before I iterate the design, consider for a moment the kinds of issues raised. When a UI becomes asynchronous, it forces you to think about additional states in your UI. I recommend you ask yourself at least these questions:

1. How will the UI display errors? (I hope your synchronous UI already has an answer for this one!)
2. How should the UI look while the operation is in progress? (For example, will it provide immediate feedback via busy indicators?)
3. How is the user restricted while the operation is in progress? (Are buttons disabled, for example?)
4. Does the user have any additional commands available while the operation is in progress? (For instance, can he cancel the operation?)
5. If the user can start multiple operations, how does the UI provide completion or error details for each one? (For example, will the UI use a "command queue" style or notification popups?)

Handling Asynchronous Command Completion via Data Binding

Most of the problems in the first AsyncCommand iteration relate to how the results are handled. What's really needed is some kind of type that would wrap a Task<T> and provide



http://www.example.com/

1270

Only 'http' and 'https' schemes are allowed.
Parameter name: requestUri

Canceled

Loading (http://www.example.com/)...

Figure 2 **An Application Executing Multiple Commands**

some data-binding capabilities so the application can respond more elegantly. As it happens, the NotifyTaskCompletion<T> type developed in my last article fits these needs almost perfectly. I'm going to add one member to this type that simplifies some of the AsyncCommand logic: a TaskCompletion property that represents the operation completing but doesn't propagate exceptions (or return a result). Here are the modifications to NotifyTaskCompletion<T>:

```
public NotifyTaskCompletion(Task<TResult> task)
{
  Task = task;
  if (!task.IsCompleted)
    TaskCompletion = WatchTaskAsync(task);
}
public Task TaskCompletion { get; private set; }
```

The next iteration of AsyncCommand uses NotifyTaskCompletion to represent the actual operation. By doing so, the XAML can data bind directly to the result and error message of that operation, and it can also use data binding to display an appropriate message while the operation is in progress. The new AsyncCommand now has a property that represents the actual operation, as shown in **Figure 7**.

Note that AsyncCommand.ExecuteAsync is using TaskCompletion and not Task. I don't want to propagate exceptions to the UI main loop (which would happen if it awaited the Task property); instead, I return TaskCompletion and handle exceptions by data binding. I also added a simple NullToVisibilityConverter to the project so that the busy indicator, results and error message are all hidden until the button is clicked. **Figure 8** shows the updated ViewModel code.

And the new XAML code is shown in **Figure 9**.

> Cancellation itself is always a synchronous operation—the act of requesting cancellation is immediate.

The code now matches the AsyncCommands2 project in the sample code. This code takes care of all the concerns I mentioned with the original solution: labels are hidden until the first operation starts; there's an immediate busy indicator providing feedback to the user; exceptions are captured and update the UI via data binding; multiple requests no longer interfere with each other. Each request creates a new NotifyTaskCompletion wrapper, which has its own independent Result and other properties. NotifyTaskCompletion acts as a data-bindable abstraction of an asynchronous operation. This allows multiple requests, with the UI always binding to the latest request. However, in many real-world scenarios, the appropriate solution is to disable multiple requests. That is, you want the command to return false from CanExecute while there's an operation in progress. This is easy enough to do with a small modification to AsyncCommand, as shown in **Figure 10**.

Figure 3 **The Service Layer**

```
public static class MyService
{
  // bit.ly/1fCnbJ2
  public static async Task<int> DownloadAndCountBytesAsync(string url,
    CancellationToken token = new CancellationToken())
  {
    await Task.Delay(TimeSpan.FromSeconds(3), token).ConfigureAwait(false);
    var client = new HttpClient();
    using (var response = await client.GetAsync(url, token).ConfigureAwait(false))
    {
      var data = await
        response.Content.ReadAsByteArrayAsync().ConfigureAwait(false);
      return data.Length;
    }
  }
}
```

Figure 4 **Base Type for Asynchronous Commands**

```
public abstract class AsyncCommandBase : IAsyncCommand
{
  public abstract bool CanExecute(object parameter);

  public abstract Task ExecuteAsync(object parameter);

  public async void Execute(object parameter)
  {
    await ExecuteAsync(parameter);
  }

  public event EventHandler CanExecuteChanged
  {
    add { CommandManager.RequerySuggested += value; }
    remove { CommandManager.RequerySuggested -= value; }
  }

  protected void RaiseCanExecuteChanged()
  {
    CommandManager.InvalidateRequerySuggested();
  }
}
```

Figure 5 **The First Attempt at an Asynchronous Command**

```
public class AsyncCommand : AsyncCommandBase
{
  private readonly Func<Task> _command;

  public AsyncCommand(Func<Task> command)
  {
    _command = command;
  }

  public override bool CanExecute(object parameter)
  {
    return true;
  }

  public override Task ExecuteAsync(object parameter)
  {
    return _command();
  }
}
```

Figure 6 **The First MainWindowViewModel**

```
public sealed class MainWindowViewModel : INotifyPropertyChanged
{
  public MainWindowViewModel()
  {
    Url = "http://www.example.com/";
    CountUrlBytesCommand = new AsyncCommand(async () =>
    {
      ByteCount = await MyService.DownloadAndCountBytesAsync(Url);
    });
  }

  public string Url { get; set; } // Raises PropertyChanged
  public IAsyncCommand CountUrlBytesCommand { get; private set; }
  public int ByteCount { get; private set; } // Raises PropertyChanged
}
```

Now the code matches the AsyncCommands3 project in the sample code. The button is disabled while the operation is going on.

Adding Cancellation

Many asynchronous operations can take varying amounts of time. For example, an HTTP request may normally respond very quickly, before the user can even respond. However, if the network is slow or the server is busy, that same HTTP request might cause a considerable delay. Part of designing an asynchronous UI is expecting and designing for this scenario. The current solution already has a busy indicator. When you design an asynchronous UI, you can also choose to give the user more options, and cancellation is a common choice.

Cancellation itself is always a synchronous operation—the act of requesting cancellation is immediate. The trickiest part of cancellation is when it can run; it should be able to execute only when there's an asynchronous command in progress. The modifications to AsyncCommand in **Figure 11** provide a nested cancellation command and notify that cancellation command when the asynchronous command begins and ends.

Adding a Cancel button (and a canceled label) to the UI is straightforward, as **Figure 12** shows.

In my opinion, as a community we haven't come up with a really good UX for handling multiple asynchronous operations.

Now, if you execute the application (AsyncCommands4 in the sample code), you'll find the cancel button is initially disabled. It's enabled when you click the Go button and remains enabled until the operation completes (whether successfully, faulted or canceled). You now have an arguably complete UI for an asynchronous operation.

A Simple Work Queue

Up to this point, I've been focusing on a UI for just one operation at a time. This is all that's necessary in many situations, but sometimes you need the ability to start multiple asynchronous operations. In my opinion, as a community we haven't come up with a really good UX for handling multiple asynchronous operations. Two common approaches are using a work queue or a notification system, neither of which is ideal.

The trickiest part of cancellation is when it can run; it should be able to execute only when there's an asynchronous command in progress.

A work queue displays all asynchronous operations in a collection; this gives the user maximum visibility and control, but is usually too complex for the typical end user to cope with. A notification system hides the operations while they're running, and will pop up if any of them fault (and possibly if they complete successfully). A notification system is more user-friendly, but it doesn't provide the full visibility and power of the work queue (for example, it's difficult to work cancellation into a notification-based system). I have yet to discover an ideal UX for multiple asynchronous operations.

Figure 7 **The Second Attempt at an Asynchronous Command**

```
public class AsyncCommand<TResult> : AsyncCommandBase, INotifyPropertyChanged
{
  private readonly Func<Task<TResult>> _command;
  private NotifyTaskCompletion<TResult> _execution;

  public AsyncCommand(Func<Task<TResult>> command)
  {
    _command = command;
  }

  public override bool CanExecute(object parameter)
  {
    return true;
  }

  public override Task ExecuteAsync(object parameter)
  {
    Execution = new NotifyTaskCompletion<TResult>(_command());
    return Execution.TaskCompletion;
  }

  // Raises PropertyChanged
  public NotifyTaskCompletion<TResult> Execution { get; private set; }
}
```

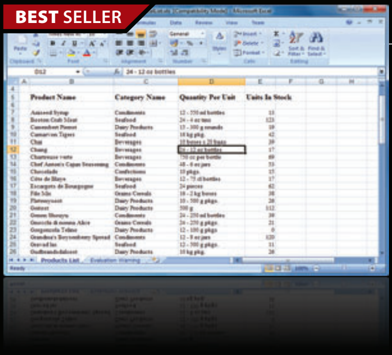
Figure 8 **The Second MainWindowViewModel**

```
public sealed class MainWindowViewModel : INotifyPropertyChanged
{
  public MainWindowViewModel()
  {
    Url = "http://www.example.com/";
    CountUrlBytesCommand = new AsyncCommand<int>(() => MyService.
DownloadAndCountBytesAsync(Url));
  }

  // Raises PropertyChanged
  public string Url { get; set; }

  public IAsyncCommand CountUrlBytesCommand { get; private set; }
}
```

/update/2014/04

ComponentSource®
The Definitive Source of Software Components
www.componentsource.com

BEST SELLER



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

ASPOSE
Your File Format APIs

Every Aspose .NET component in one package.

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

BEST SELLER

GdPicture.NET Ultimate | from $4,127.59

GdPicture
imaging technologies

All-in-one AnyCPU document-imaging and PDF toolkit for .NET and ActiveX.

- Document viewing, processing, printing, scanning, OMR, OCR, Barcode Recognition, DICOM
- Annotate image and PDF within your Windows & Web applications
- Read, write and convert vector & raster images in more than 90 formats, including PDF
- Color detection engine for image and PDF compression
- 100% royalty-free and world leading Imaging SDK

BEST SELLER

ComponentOne Studio Enterprise 2014 v1 | from $1,315.60

ComponentOne
a division of GrapeCity

.NET Tools for the Professional Developer: Windows, HTML5/Web, and XAML.

- Hundreds of UI controls for all .NET platforms including grids, charts, reports and schedulers
- Visual Studio 2013 and Bootstrap support
- Advanced theming tools for WinForms and ASP.NET
- 40+ UI widgets built with HTML5, jQuery, CSS3, and SVG
- Windows Store Sparkline, DropDown, & Excel controls

BEST SELLER

Help & Manual Professional | from $583.10

ec software

Easily create documentation for Windows, the Web and iPad.

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

We accept purchase orders.
Contact us to apply for a credit account.

US Headquarters
ComponentSource
650 Claremore Prof Way
Suite 100
Woodstock
GA 30188-5188
USA

European Headquarters
ComponentSource
30 Greyfriars Road
Reading
Berkshire
RG1 1PE
United Kingdom

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

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

MasterCard VISA DISCOVER

GSA Schedule
Contract GS-35F-0188R

That said, the sample code at this point can be extended to support a multiple-operation scenario without too much trouble. In the existing code, the Go button and the Cancel button are both conceptually related to a single asynchronous operation. The new UI will change the Go button to mean "start a new asynchronous operation and add it to the list of operations." What this means is that the Go button is now actually synchronous. I added a simple (synchronous) DelegateCommand to the solution, and now the ViewModel and XAML can be updated, as **Figure 13** and **Figure 14** show.

There isn't a universal solution for an asynchronous command that fits everyone's needs—yet.

This code is equivalent to the AsyncCommandsWithQueue project in the sample code. When the user clicks the Go button, a new AsyncCommand is created and wrapped into a child ViewModel (CountUrlBytesViewModel). This child ViewModel instance is then added to the list of operations. Everything associated with that particular operation (the various labels and the Cancel button) is displayed in a data template for the work queue. I also added a simple button "X" that will remove the item from the queue.

Figure 9 The Second MainWindow XAML

```
<Grid>
  <TextBox Text="{Binding Url}" />
  <Button Command="{Binding CountUrlBytesCommand}" Content="Go" />
  <Grid Visibility="{Binding CountUrlBytesCommand.Execution,
    Converter={StaticResource NullToVisibilityConverter}}">
    <!--Busy indicator-->
    <Label Visibility="{Binding CountUrlBytesCommand.Execution.IsNotCompleted,
      Converter={StaticResource BooleanToVisibilityConverter}}"
      Content="Loading..." />
    <!--Results-->
    <Label Content="{Binding CountUrlBytesCommand.Execution.Result}"
      Visibility="{Binding CountUrlBytesCommand.Execution.IsSuccessfullyCompleted,
      Converter={StaticResource BooleanToVisibilityConverter}}" />
    <!--Error details-->
    <Label Content="{Binding CountUrlBytesCommand.Execution.ErrorMessage}"
      Visibility="{Binding CountUrlBytesCommand.Execution.IsFaulted,
      Converter={StaticResource BooleanToVisibilityConverter}}" Foreground="Red" />
  </Grid>
</Grid>
```

Figure 10 Disabling Multiple Requests

```
public class AsyncCommand<TResult> : AsyncCommandBase, INotifyPropertyChanged
{
  public override bool CanExecute(object parameter)
  {
    return Execution == null || Execution.IsCompleted;
  }

  public override async Task ExecuteAsync(object parameter)
  {
    Execution = new NotifyTaskCompletion<TResult>(_command());
    RaiseCanExecuteChanged();
    await Execution.TaskCompletion;
    RaiseCanExecuteChanged();
  }
}
```

This is a very basic work queue, and I made some assumptions about the design. For example, when an operation is removed from the queue, it isn't automatically canceled. When you start working with multiple asynchronous operations, I recommend you ask yourself at least these additional questions:

1. How does the user know which notification or work item is for which operation? (For example, the busy indicator in this work queue sample contains the URL it's downloading).
2. Does the user need to know every result? (For example, it may be acceptable to notify the user only of errors, or to automatically remove successful operations from the work queue).

Figure 11 Adding Cancellation

```
public class AsyncCommand<TResult> : AsyncCommandBase, INotifyPropertyChanged
{
  private readonly Func<CancellationToken, Task<TResult>> _command;
  private readonly CancelAsyncCommand _cancelCommand;
  private NotifyTaskCompletion<TResult> _execution;

  public AsyncCommand(Func<CancellationToken, Task<TResult>> command)
  {
    _command = command;
    _cancelCommand = new CancelAsyncCommand();
  }

  public override async Task ExecuteAsync(object parameter)
  {
    _cancelCommand.NotifyCommandStarting();
    Execution = new NotifyTaskCompletion<TResult>(_command(_cancelCommand.Token));
    RaiseCanExecuteChanged();
    await Execution.TaskCompletion;
    _cancelCommand.NotifyCommandFinished();
    RaiseCanExecuteChanged();
  }

  public ICommand CancelCommand
  {
    get { return _cancelCommand; }
  }

  private sealed class CancelAsyncCommand : ICommand
  {
    private CancellationTokenSource _cts = new CancellationTokenSource();
    private bool _commandExecuting;

    public CancellationToken Token { get { return _cts.Token; } }

    public void NotifyCommandStarting()
    {
      _commandExecuting = true;
      if (!_cts.IsCancellationRequested)
        return;
      _cts = new CancellationTokenSource();
      RaiseCanExecuteChanged();
    }

    public void NotifyCommandFinished()
    {
      _commandExecuting = false;
      RaiseCanExecuteChanged();
    }

    bool ICommand.CanExecute(object parameter)
    {
      return _commandExecuting && !_cts.IsCancellationRequested;
    }

    void ICommand.Execute(object parameter)
    {
      _cts.Cancel();
      RaiseCanExecuteChanged();
    }
  }
}
```

Wrapping Up

There isn't a universal solution for an asynchronous command that fits everyone's needs—yet. The developer community is still exploring asynchronous UI patterns. My goal in this article is to show how to think about asynchronous commands in the context of an MVVM application, especially considering UX issues that must be addressed when the UI becomes asynchronous. But keep in mind the patterns in this article and sample code are just patterns, and should be adapted to the needs of the application.

Figure 12 **Adding a Cancel Button**

```
<Grid>
  <TextBox Text="{Binding Url}" />
  <Button Command="{Binding CountUrlBytesCommand}" Content="Go" />
  <Button Command="{Binding CountUrlBytesCommand.CancelCommand}" Content="Cancel" />
  <Grid Visibility="{Binding CountUrlBytesCommand.Execution,
    Converter={StaticResource NullToVisibilityConverter}}">
    <!--Busy indicator-->
    <Label Content="Loading..."
      Visibility="{Binding CountUrlBytesCommand.Execution.IsNotCompleted,
      Converter={StaticResource BooleanToVisibilityConverter}}" />
    <!--Results-->
    <Label Content="{Binding CountUrlBytesCommand.Execution.Result}"
      Visibility="{Binding CountUrlBytesCommand.Execution.IsSuccessfullyCompleted,
      Converter={StaticResource BooleanToVisibilityConverter}}" />
    <!--Error details-->
    <Label Content="{Binding CountUrlBytesCommand.Execution.ErrorMessage}"
      Visibility="{Binding CountUrlBytesCommand.Execution.IsFaulted,
      Converter={StaticResource BooleanToVisibilityConverter}}" Foreground="Red" />
    <!--Canceled-->
    <Label Content="Canceled"
      Visibility="{Binding CountUrlBytesCommand.Execution.IsCanceled,
      Converter={StaticResource BooleanToVisibilityConverter}}" Foreground="Blue" />
  </Grid>
</Grid>
```

Figure 13 **ViewModel for Multiple Commands**

```
public sealed class CountUrlBytesViewModel
{
  public CountUrlBytesViewModel(MainWindowViewModel parent, string url,
    IAsyncCommand command)
  {
    LoadingMessage = "Loading (" + url + ")...";
    Command = command;
    RemoveCommand = new DelegateCommand(() => parent.Operations.Remove(this));
  }

  public string LoadingMessage { get; private set; }

  public IAsyncCommand Command { get; private set; }

  public ICommand RemoveCommand { get; private set; }
}

public sealed class MainWindowViewModel : INotifyPropertyChanged
{
  public MainWindowViewModel()
  {
    Url = "http://www.example.com/";
    Operations = new ObservableCollection<CountUrlBytesViewModel>();
    CountUrlBytesCommand = new DelegateCommand(() =>
    {
      var countBytes = new AsyncCommand<int>(token =>
        MyService.DownloadAndCountBytesAsync(
        Url, token));
      countBytes.Execute(null);
      Operations.Add(new CountUrlBytesViewModel(this, Url, countBytes));
    });
  }

  public string Url { get; set; } // Raises PropertyChanged

  public ObservableCollection<CountUrlBytesViewModel> Operations
    { get; private set; }

  public ICommand CountUrlBytesCommand { get; private set; }
}
```

Figure 14 **XAML for Multiple Commands**

```
<Grid>
  <TextBox Text="{Binding Url}" />
  <Button Command="{Binding CountUrlBytesCommand}" Content="Go" />
  <ItemsControl ItemsSource="{Binding Operations}">
    <ItemsControl.ItemTemplate>
      <DataTemplate>
        <Grid>
          <!--Busy indicator-->
          <Label Content="{Binding LoadingMessage}"
            Visibility="{Binding Command.Execution.IsNotCompleted,
            Converter={StaticResource BooleanToVisibilityConverter}}" />
          <!--Results-->
          <Label Content="{Binding Command.Execution.Result}"
            Visibility="{Binding Command.Execution.IsSuccessfullyCompleted,
            Converter={StaticResource BooleanToVisibilityConverter}}" />
          <!--Error details-->
          <Label Content="{Binding Command.Execution.ErrorMessage}"
            Visibility="{Binding Command.Execution.IsFaulted,
            Converter={StaticResource BooleanToVisibilityConverter}}"
            Foreground="Red" />
          <!--Canceled-->
          <Label Content="Canceled"
            Visibility="{Binding Command.Execution.IsCanceled,
            Converter={StaticResource BooleanToVisibilityConverter}}"
            Foreground="Blue" />
          <Button Command="{Binding Command.CancelCommand}" Content="Cancel" />
          <Button Command="{Binding RemoveCommand}" Content="X" />
        </Grid>
      </DataTemplate>
    </ItemsControl.ItemTemplate>
  </ItemsControl>
</Grid>
```

In particular, there isn't a perfect story regarding multiple asynchronous operations. There are drawbacks to both work queues and notifications, and it seems to me that a universal UX has yet to be developed. As more UIs become asynchronous, a lot more minds will be thinking about this problem, and a revolutionary breakthrough might be right around the corner. Give the problem some thought, dear reader. Perhaps you will be the discoverer of a new UX.

In the meantime, you still have to ship. In this article I started with the most basic of asynchronous ICommand implementations and gradually added features until I ended up with something fairly suitable for most modern applications. The result is also fully unit-testable; because the async void ICommand.Execute method *only* calls the Task-returning IAsyncCommand.ExecuteAsync method, you can use ExecuteAsync directly in your unit tests.

In my last article, I developed NotifyTaskCompletion<T>, a data-binding wrapper around Task<T>. In this one, I showed how to develop one kind of AsyncCommand<T>, an asynchronous implementation of ICommand. In my next article, I'll address asynchronous services. Do bear in mind that asynchronous MVVM patterns are still quite new; don't be afraid to deviate from them and innovate your own solutions. ■

Stephen Cleary *is a husband, father and programmer living in northern Michigan. He has worked with multithreading and asynchronous programming for 16 years and has used async support in the Microsoft .NET Framework since the first CTP. His homepage, including his blog, is at stephencleary.com.*

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

Build a Cross-Platform, Mobile Golf App Using C# and Xamarin

Wallace B. McClure

One of the fun things about the return of golf season is participating in tournaments that feature events such as a longest drive contest. In these, a person's first shot off a designated hole is measured against others in the tournament. The longest drive during the day is declared the winner. However, these contests typically don't have centralized scoring. If you're in the first group, you don't know until after the event is over where your shots stand in relation to everyone else's. Why not use a mobile phone to record the starting point and ending point of drives and store the information in a cloud-hosted database?

This article discusses:

- Technology choices to build an example app
- Using Windows Azure Mobile Services
- Scaling in Windows Azure
- Using Windows Azure SQL Database
- Efficient data access
- Sharing code
- Push notifications

Technologies discussed:

Windows Phone, C#, Microsoft .NET Framework, Apple iOS, Xamarin, Windows Azure Mobile Services

Code download available at:

msdn.microsoft.com/magazine/msdnmag0414

The options for building such an app are many, which can be confusing. In this article, I'll walk through how I built such an app using the back-end options in Windows Azure and how I handled various issues. I'll show the code for writing an app for Windows Phone as well as iOS using Xamarin.

Several features were required. The app needed to run across mobile devices and multiple device OSes. It had to be a native app that looked just like all of the others on a device. The back-end server had to be always available, with minimal hassle to the developer (me). The cloud services had to provide as much help as possible in the area of cross-platform development. The back-end database needed to provide some amount of geolocation functionality.

Why C#?

Options to build cross-platform apps include: mobile Web apps; Xamarin C# for the iPhone (and Android); forgoing cross-platform and building apps in the vendor-directed language (the Microsoft .NET Framework for Windows Phone and Objective-C for the iPhone); and several others. First off, the choice of C#/.NET Framework as the language on the client made sense to me. Having a background in C# meant after learning the platform-specific features of a device there was one less item to spend time learning. Being able to develop everything in Visual Studio 2013 was an even bigger reason to go with a Xamarin solution for the iPhone. The problem with the mobile Web option is users want apps that integrate as deeply with their platform as possible. This is difficult with a mobile Web solution, but it's easier with a native solution. Forgoing cross-platform and going with a vendor-directed solution didn't make sense because it would require learning a new language for each platform.

Development Tools

Building an app for multiple platforms has typically required multiple development tools. Before the advent of Xamarin.iOS, development for the iPhone could only be done on a Mac using Xamarin Studio (previously MonoDevelop, which was a port of the open source SharpDevelop). While there's nothing wrong with Xamarin Studio, developers typically want to stay within the same IDE as much as possible. Visual Studio 2013 used with Xamarin.iOS for Visual Studio lets you develop for Windows Azure, Windows Phone and the iPhone without having to leave the IDE you know and love.

Windows Azure

A mobile device can interact with Windows Azure in several ways. These include Windows Azure Virtual Machine (VM hereafter for brevity), Web Role, Windows Azure Web Sites and Windows Azure Mobile Services (WAMS).

A VM provides the most control of all the variables. You can make changes to your app and all of the server's settings. This is great for those apps that require customization to the underlying OS settings, another app to be installed or other possible changes. This is referred to as Infrastructure as a Service (IaaS).

> Being able to develop everything in Visual Studio 2013 was an even bigger reason to go with a Xamarin solution for the iPhone.

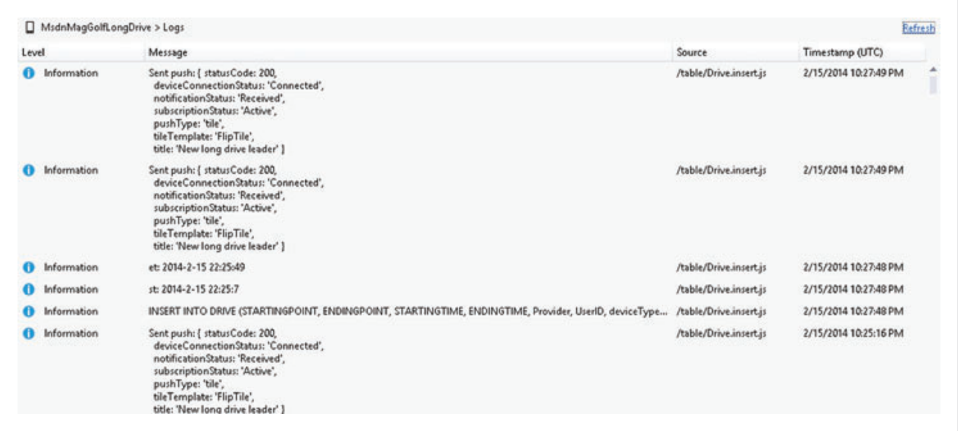
Windows Azure has a Cloud Service project type that encompasses a set of roles. Roles typically map to projects in Visual Studio. A Web Role is basically a Web project that has been bundled in a single deployable package that's uploaded and runs within a VM. A Web Role provides Web UIs and can also have Web services within it. A Worker Role is a project that will continually run on the server. This is referred to as Platform as a Service (PaaS).

Windows Azure Web Sites are similar in concept to a Web Role. This solution will let an app host a Web site or project. The project can include Web services. These Web services can be called via a SOAP or REST call. Windows Azure Web Sites are a great solution for when an application only needs IIS.

The first three options require you to have complete knowledge of Web services—how to call them, storing them in the database, and the plumbing that's in between the mobile device and the cloud. Microsoft has a solution that lets you quickly and easily store data in the cloud, handle push notifications and authenticate users quickly and easily: WAMS. Having built solutions with the other

Figure 1 **The Insert.js File Used When a Golf Drive Is Inserted into the Cloud**

```
function insert(item, user, request) {
  if ((!isNaN(item.StartingLat)) && (!isNaN(item.StartingLon)) &&
    (!isNaN(item.EndingLat)) && (!isNaN(item.EndingLon))) {
    var distance1 = 0.0;
    var distance2 = 0.0;
    var sd = item.StartingTime;
    var ed = item.EndingTime;
    var sdate = new Date(sd);
    var edate = new Date(ed);
    var res = user.userId.split(":");
    var provider = res[0].replace("'", "''");
    var userId = res[1].replace("'", "''");
    var insertStartingDate = sdate.getFullYear() + "-" +
      (sdate.getMonth() + 1) + "-" + sdate.getDate() + " " +
      sdate.getHours() + ":" + sdate.getMinutes() + ":" + sdate.getSeconds();
    var insertEndingDate = edate.getFullYear() + "-" +
      (edate.getMonth() + 1) + "-" + edate.getDate() + " " +
      edate.getHours() + ":" + edate.getMinutes() + ":" + edate.getSeconds();
    var lat1 = item.StartingLat;
    var lon1 = item.StartingLon;
    var lat2 = item.EndingLat;
    var lon2 = item.EndingLon;
    var sp = "'POINT(" + item.StartingLon + " " + item.StartingLat + ")'";
    var ep = "'POINT(" + item.EndingLon + " " + item.EndingLat + ")'";
    var sql = "select Max(Distance) as LongDrive from Drive";
    mssql.query(sql, [], {
      success: function (results) {
        if ( results.length == 1)
        {
          distance1 = results[0].LongDrive;
        }
      }
    });

    var sqlDis = "select [dbo].[CalculateDistanceViaLatLon](?, ?, ?, ?)";
    var args = [lat1, lon1, lat2, lon2];
    mssql.query(sqlDis, args, {
      success: function (distance) {
        distance2 = distance[0].Column0;
      }
    });

    var queryString = "INSERT INTO DRIVE (STARTINGPOINT, ENDINGPOINT, " +
      "STARTINGTIME, ENDINGTIME, Provider, UserID, " +
      "deviceType, deviceToken, chanelUri) VALUES " +
      "(geography::STPointFromText(" + sp + ", 4326), " +
      " geography::STPointFromText(" + ep + ", 4326), " +
      " '" + insertStartingDate + "', '" +
      insertEndingDate + "', '" + provider + "', " + userId + ", " +
      item.deviceType + ", '" + item.deviceToken.replace("'", "''") +
      "', " + "'" + item.ChannelUri.replace("'", "''") + "')";
    console.log(queryString);
    mssql.query(queryString, [], {
      success: function () {
        if (distance2 > distance1) {
          if (item.deviceType == 0) {
            push.mpns.sendFlipTile(item.ChannelUri, {
              title: "New long drive leader"
            }, {
                  success: function (pushResponse) {
                    console.log("Sent push:", pushResponse);
                  }
               });
            }
          if (item.deviceType == 1) {
            push.apns.send(item.deviceToken, {
              alert: "New Long Drive",
              payload: {
                inAppMessage: "Hey, there is now a new long drive."
              }
            });
          }
        }
      },
      error: function (err) {
        console.log("Error: " + err);
      }
    });
    request.respond(200, {});
  }
}
```



MsdnMagGolfLongDrive > Logs

Refresh

Level	Message	Source	Timestamp (UTC)
Information	Sent push: { statusCode: 200, deviceConnectionStatus: 'Connected', notificationStatus: 'Received', subscriptionStatus: 'Active', pushType: 'tile', tileTemplate: 'FlipTile', title: 'New long drive leader' }	/table/Drive.insert.js	2/15/2014 10:27:49 PM
Information	Sent push: { statusCode: 200, deviceConnectionStatus: 'Connected', notificationStatus: 'Received', subscriptionStatus: 'Active', pushType: 'tile', tileTemplate: 'FlipTile', title: 'New long drive leader' }	/table/Drive.insert.js	2/15/2014 10:27:49 PM
Information	et: 2014-2-15 22:25:49	/table/Drive.insert.js	2/15/2014 10:27:48 PM
Information	st: 2014-2-15 22:25:7	/table/Drive.insert.js	2/15/2014 10:27:48 PM
Information	INSERT INTO DRIVE (STARTINGPOINT, ENDINGPOINT, STARTINGTIME, ENDINGTIME, Provider, UserID, deviceType...	/table/Drive.insert.js	2/15/2014 10:27:48 PM
Information	Sent push: { statusCode: 200, deviceConnectionStatus: 'Connected', notificationStatus: 'Received', subscriptionStatus: 'Active', pushType: 'tile', tileTemplate: 'FlipTile', title: 'New long drive leader' }	/table/Drive.insert.js	2/15/2014 10:25:16 PM

Figure 2 **Log File Information in Visual Studio 2013**

options, I thought it sensible to use WAMS, given the need for the least amount of plumbing as possible and its good cross-platform support. WAMS is sometimes described as the "back end that you don't have to build."

Windows Azure Mobile Services

WAMS lets you accelerate your mobile development efforts by providing storage, user authentication against multiple social networks, mechanisms to create logic on the server (with Node.js), push notifications, and a packaged client-side code library to ease create, read, update and delete (CRUD) operations against the data.

The first step in using WAMS is to create the mobile service and an associated database table. The database table isn't strictly required. For more details on the setup process, see the Windows Azure tutorial at bit.ly/Nc8rWX.

Server Scripts Basic CRUD operations are available in WAMS via server operations. These are the delete.js, insert.js, read.js and update.js files, which are processed via Node.js on the server. For more information on Node.js in WAMS, see the article, "Work with server scripts in Mobile Services," on the Windows Azure site at bit.ly/1cHASFA.

Let's start by taking a look at the insert.js file in **Figure 1**. In the method signature, the "item" parameter contains the data that's handed in. The members of the object map to the data object handed in from the client. The members of this object will make more sense after looking at the section on filling data from the client. The "user" parameter contains information regarding the connected user. In this example, the user must be authenticated. The app uses Facebook and Twitter for authentication, so the userId that's returned has the form of "Network:12345678." The "Network" part of the value contains the name of the network provider. In this example, either Facebook or Twitter is available, so one of these will be a part of the value. The number "12345678" is merely a representation of the userId. While Twitter and Facebook are used in this example, Windows Azure can also use a Microsoft or Google account.

The first thing to do is test the code to validate the input. I want to verify that the latitudes and longitudes brought in are valid numbers. If not, the insert will exit immediately. The next step is to parse the passed-in userId to get the network provider and the numeric user identifier. The third step is to set up the dates so they can be inserted into the database. JavaScript and SQL Server have mismatched date/time representations, so these must be parsed and placed in the correct format.

Now a query needs to be done. A Node.js command to perform a CRUD statement calls mssql.query(command, parameters, callbacks). The "command" parameter is the SQL command that will be executed. The "parameters" parameter is a JavaScript array that matches up with the parameters specified in the command that's set. The "callbacks" parameter contains the JavaScript callbacks to be used when the query is completed, depending on success or

Figure 3 **SQL Function to Calculate Distance Between Two Points**

```
CREATE FUNCTION [dbo].[CalculateDistanceViaLatLon]
(
  @lat1 float,
  @lon1 float,
  @lat2 float,
  @lon2 float
)
RETURNS float
AS
BEGIN
  declare @g1 sys.geography = sys.geography::Point(@lat1, @lon1, 4326)
  declare @g2 sys.geography = sys.geography::Point(@lat2, @lon2, 4326)
  RETURN @g1.STDistance(@g2)
END
CREATE FUNCTION [dbo].[CalculateDistance]
(
  @param1 [sys].[geography],
  @param2 [sys].[geography]
)
RETURNS INT
AS
BEGIN
  RETURN @param1.STDistance(@param2)
END
```

Figure 4 **SQL Table to Hold Drive Data**

```
CREATE TABLE [MsdnMagGolfLongDrive].[Drive] (
  [id]             NVARCHAR (255)     CONSTRAINT [DF_Drive_id] DEFAULT
(CONVERT([nvarchar](255),newid(),(0))) NOT NULL,
  [__createdAt]    DATETIMEOFFSET (3) CONSTRAINT
    [DF_Drive___createdAt] DEFAULT (CONVERT([datetimeoffset](3),
    sysutcdatetime(),(0))) NOT NULL,
  [__updatedAt]    DATETIMEOFFSET (3) NULL,
  [__version]      ROWVERSION         NOT NULL,
  [UserID]         BIGINT             NULL,
  [StartingPoint] [sys].[geography]  NULL,
  [EndingPoint]    [sys].[geography]  NULL,
  [DateEntered]    DATETIME           NULL,
  [DateUpdated]    DATETIME           NULL,
  [StartingTime]   DATETIME           NULL,
  [EndingTime]     DATETIME           NULL,
  [Distance]       AS                 ([dbo].[CalculateDistance]
    ([StartingPoint],[EndingPoint])),
  [Provider]       NVARCHAR (20)      NULL,
  [deviceToken]    NVARCHAR (100)     NULL,
  [deviceType] INT NULL,

  PRIMARY KEY NONCLUSTERED ([id] ASC)
);
```

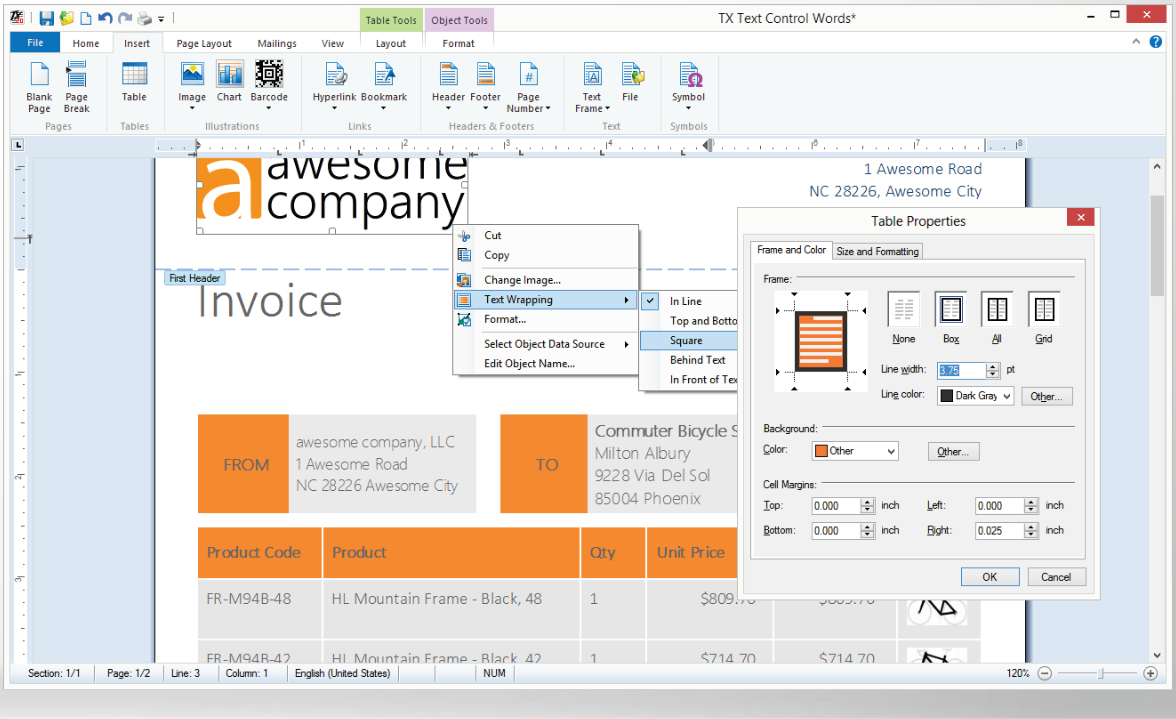
TX TEXT CONTROL®

word processing components

There are many .NET 'Rich Text Editors' out there...

Meh, sorry!

There is only one



awesome company

1 Awesome Road
NC 28226, Awesome City

Invoice

FROM	awesome company, LLC 1 Awesome Road NC 28226 Awesome City	TO	Commuter Bicycle S Milton Albury 9228 Via Del Sol 85004 Phoenix

Product Code	Product	Qty	Unit Price		
FR-M94B-48	HL Mountain Frame - Black, 48	1	$809.76	$809.76	
FR-M94B-42	HL Mountain Frame - Black, 42	1	$714.70	$714.70	

Table Properties

Frame and Color | Size and Formatting

Frame: None / Box / All / Grid

Line width: 3.75 pt

Line color: Dark Gray — Other...

Background:

Color: Other — Other...

Cell Margins:

Top: 0.000 inch — Left: 0.000 inch

Bottom: 0.000 inch — Right: 0.025 inch

Section: 1/1 | Page: 1/2 | Line: 3 | Column: 1 | English (United States) | NUM

Reporting ▪ Rich Text Editing ▪ Spell Checking ▪ Barcodes

www.textcontrol.com

/txtextcontrol

US: +1 855-533-TEXT
EU: +49 421 427 067-10

Visual Studio Partner

Figure 5 **A Proxy to Work with REST**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Newtonsoft.Json;

namespace Support
{
  public partial class Drive
  {
    public Drive() {
      DeviceToken = String.Empty;
      ChannelUri = String.Empty;
    }
    [JsonProperty(PropertyName="id")]
    public string Id { get; set; }
    [JsonProperty(PropertyName = "UserID")]
    public Int64 UserID { get; set; }
    [JsonProperty(PropertyName = "Provider")]
    public string Provider { get; set; }
    public double StartingLat { get; set; }
    public double StartingLon { get; set; }
    public double EndingLat { get; set; }
    public double EndingLon { get; set; }
    [JsonProperty(PropertyName = "StartingTime")]
    public DateTime StartingTime { get; set; }
    [JsonProperty(PropertyName = "EndingTime")]
    public DateTime EndingTime { get; set; }
    [JsonProperty(PropertyName = "Distance")]
    public double Distance { get; set; }
    [JsonProperty(PropertyName = "deviceType")]
    public int deviceType { get; set; }
    [JsonProperty(PropertyName = "deviceToken")]
    public string DeviceToken { get; set; }
    [JsonProperty(PropertyName = "ChannelUri")]
    public string ChannelUri { get; set; }
  }
}
```

error. I'll discuss the contents of a successful initial query in the section on push notifications.

Finally, the question of debugging comes up. How do you know what's happening in the script? JavaScript has the console.log(info) method. When this method is called with the "info" parameter, the parameter is saved within the log file of the service, as shown in **Figure 2**. Note the built-in refresh functionality at the top right of the screen.

Once WAMS is set up, it can be managed via the windowsazure.com portal or Visual Studio.

Note: Calling from a WAMS script file to a method may result in an error with the default setup because they run in different schemas. Permissions might need to be granted depending on your specific situation. Jeff Sanders has a blog post about this issue at bit.ly/1cHQ4Cu.

Scale

Mobile apps can put a tremendous load on infrastructure, and luckily Windows Azure has several options to handle this. First, you have several alternatives for message queuing.

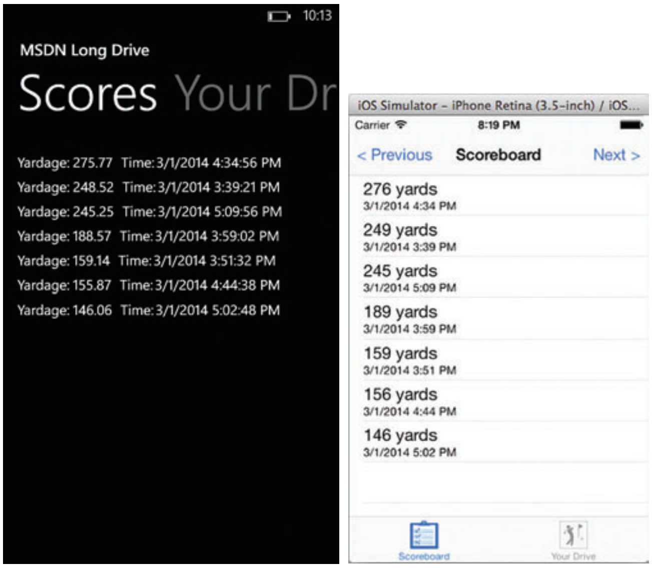
Queuing is available in Windows Azure via Service Bus as well as the Windows Azure Queue Service. Queuing lets you quickly store data without tying up the app. Under heavy load, an app can wait on a response from a remote data source. Instead of interacting directly with a data source, an app can store data in a queue, which will free the app to continue processing. Based on personal experience, using queuing can easily increase the scalability of an app. While this app doesn't use queuing, it needs to be mentioned as an option depending on the load of the operations and the number of mobile devices accessing the system. Thankfully, both the Service Bus and the Windows Azure Queue Service have the necessary APIs for the server scripts in WAMS to access each of them.

Overall, queuing is a great solution for data-intensive apps. Another tool in the scaling toolbox is auto scale. Windows Azure lets you monitor the health and availability of an app from a dashboard. You can set up rules to notify an application administrator when the availability of services is degraded. Windows Azure lets an app scale up or down to match demand. By default, this feature is turned off. When it's turned on, Windows Azure will periodically check the number of API calls on the service and will scale up if the number of calls is at or more than 90 percent of the API quota. Every day, Windows Azure will scale back down to the set minimum. The general rule is to set the daily quota to handle the expected daily traffic and allow Windows Azure to scale up as necessary. As this is being written, health, monitoring and auto scaling are available in preview.

Database

Data is the root of any app and the basis for nearly every business. You can use a hosted third-party database, a database service running with a VM, Windows Azure SQL Database and probably several other options. For the back-end database, I chose to use Windows Azure SQL Database over SQL Server running within a VM for several reasons. First, it has support for location-based services in the base product. Second, Windows Azure SQL Database is optimized for performance over a baseline installation of SQL Server on a client system. Finally, there's no ongoing management of the underlying system.

Windows Azure SQL Database has the same point and geography database types as SQL Server, so it's easy to calculate distances between two points. To make this easier, I wrote two stored procedures to calculate these distances. The SQL function



Yardage: 275.77 Time:3/1/2014 4:34:56 PM
Yardage: 248.52 Time:3/1/2014 3:39:21 PM
Yardage: 245.25 Time:3/1/2014 5:09:56 PM
Yardage: 188.57 Time:3/1/2014 3:59:02 PM
Yardage: 159.14 Time:3/1/2014 3:51:32 PM
Yardage: 155.87 Time:3/1/2014 4:44:38 PM
Yardage: 146.06 Time:3/1/2014 5:02:48 PM

276 yards
3/1/2014 4:34 PM
249 yards
3/1/2014 3:39 PM
245 yards
3/1/2014 5:09 PM
189 yards
3/1/2014 3:59 PM
159 yards
3/1/2014 3:51 PM
156 yards
3/1/2014 4:44 PM
146 yards
3/1/2014 5:02 PM

Figure 6 **The Scoreboard As Depicted on a Windows Phone-Based Device and an iPhone**

Figure 7 **Inserting Data (Windows Phone-Based Device)**

```
async void PostDrive()
{
  Drive d = new Drive();
  d.StartingLat = first.Latitude;
  d.StartingLon = first.Longitude;
  d.EndingLat = second.Latitude;
  d.EndingLon = second.Longitude;
  d.StartingTime = startingTime;
  d.EndingTime = endingTime;
  d.deviceType = (int)Support.AppConstants.DeviceType.WindowsPhone8;
  d.ChannelUri = _app.CurrentChannel.ChannelUri.ToString();

  try
  {
    await _app.client.GetTable<Support.Drive>().InsertAsync(d);
  }
  catch (System.Exception exc)
  {
    Console.WriteLine(exc.Message);
  }
}
```

CalculateDistanceViaLatLon takes the floating point values of latitude and longitude. It's designed to run within the WAMS insert.js script so it's easy to calculate the distance of the incoming drive. The result can be compared with the current maximum drive within the system. The SQL function CalculateDistance takes two geography points and calculates the distance between them, as shown in **Figure 3**. The data is stored in the Drive table as SQL Server points.

Figure 4 shows the table used to hold the data about the drive. Because the columns prefixed with "__" are columns particular to Windows Azure, every effort has been made to not use them. The columns of interest are StartingPoint, EndingPoint, Distance, deviceToken and deviceType. The StartingPoint and EndingPoint columns hold the starting and ending geographic points of a drive. The Distance column is a calculated column. It's a float that uses the CalculateDistance SQL function. The deviceToken and deviceType columns hold a token that identifies the device and the type of the device (Windows Phone-based or iPhone). The app currently only communicates back to the posting device if a new drive entered is the new leader. The deviceToken and deviceType columns could be used to communicate there's a new leader and to regularly communicate other updates to competitors.

Dynamic Schema

One of the great features of WAMS is that a database table schema is dynamic by default. It's modified based on the information sent from the mobile device to the client. As you move from development to production, this should be turned off. The last thing you want is some type of schema change to a running system due to a programming mistake. This can easily be done by going to Configure in the WAMS section of the Windows Azure Portal and turning the "dynamic schema" option to off.

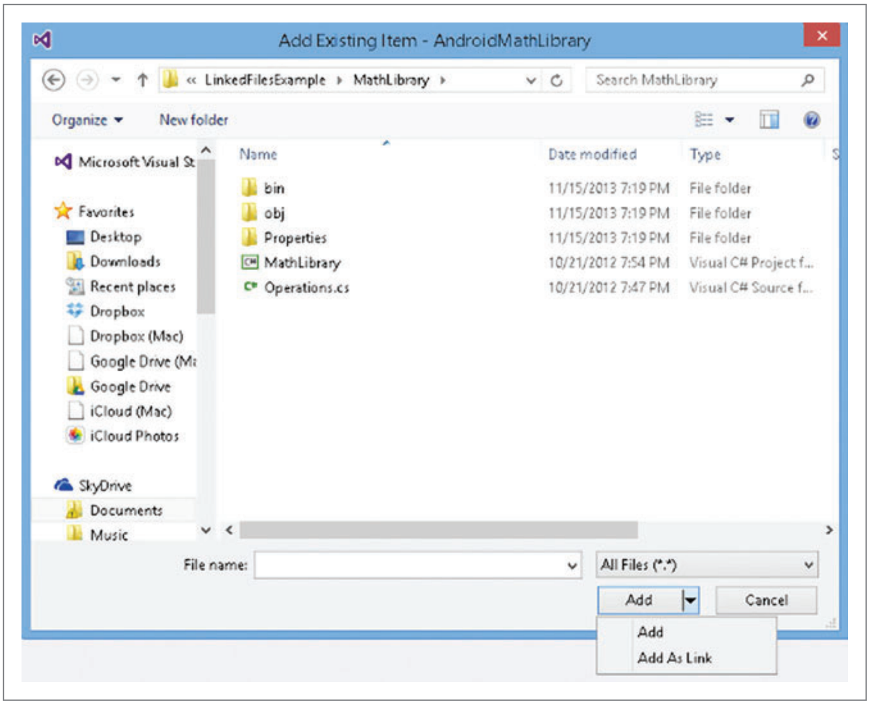
Accessing Data

Accessing data with a mobile device with an unreliable and a relatively high-latency network is significantly different from accessing data over a wired connection and relatively low-latency network. There are two general rules to get data with a device. First, data access must be done asynchronously. Given the unreliable nature and high latency of mobile networks, locking the UI thread in any way is a really bad idea. Users don't understand why the UI is stalled. If getting data takes too long, the device OS will assume the app is hung and kill it. The second rule is that the transferred data must be relatively small. Sending too many records to a mobile device will cause problems due to the low-speed, high-latency networks common in mobile provider systems and the fact that mobile device CPUs are optimized more for low power consumption than for processing data like a laptop or desktop CPU. WAMS addresses both these issues. Data access is asynchronous and queries are automatically done via a paging algorithm. To show this, I'll look at two operations, an insert and a select.

Using a Proxy Any developer who has called REST-based services knows about the problem of using REST due to its lack of a built-in proxy service. This makes working with REST somewhat mistake-prone. It's not impossible—just a little more difficult to work with than SOAP. To make development easier, you can create a proxy locally. The proxy for this example is shown in **Figure 5**. The properties of an object instance can be accessed in the server scripts.

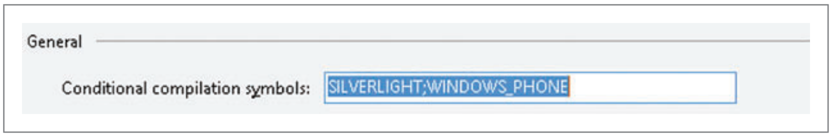
Query Data Querying data in Windows Azure is actually simple. The data will be returned via a call through a LINQ query. Here's a call to run a simple query to return the data:

```
var drives = _app.client.GetTable<Support.Drive>();
var query = drives.OrderByDescending(
  drive => drive.Distance).Skip(startingPoint).Take(PageSize);
var listedDrives = await query.ToListAsync();
```



Name	Date modified	Type
bin	11/15/2013 7:19 PM	File folder
obj	11/15/2013 7:19 PM	File folder
Properties	11/15/2013 7:19 PM	File folder
MathLibrary	10/21/2012 7:54 PM	Visual C# Project f...
Operations.cs	10/21/2012 7:47 PM	Visual C# Source f...

Figure 8 **Using Linked Files**



General

Conditional compilation symbols: SILVERLIGHT;WINDOWS_PHONE

Figure 9 **Defining a Platform as a Conditional Compilation Symbol**

In this example, I need the list of longest drives starting at the top of the list and going down. This is then bound to a grid in both a Windows Phone-based device and the iPhone. While the data binding part is different, retrieving the data is exactly same.

Note that in the preceding query, there's no Where method called, but this could easily be done. Also, the Skip and Take methods are used to show how the app could easily add in paging. **Figure 6** shows the scoreboard on a Windows Phone-based device and on an iPhone.

Inserting Data

Inserting a record is easy in WAMS. Just create an instance of the data object and then call the InsertAsync method on the client object. The code for the insert in a Windows Phone-based device is shown in **Figure 7**. The code for performing an insert in Xamarin.iOS is similar and only differs in the areas of the deviceType, ChannelUri and DeviceToken.

Sharing Code Between Platforms

Sharing code between platforms is an important consideration and can be done in several ways with the cross-platform capabilities of C# and Xamarin. The mechanism used is determined by the exact scenario. I needed to share non-UI logic. To do that, there are two options: Portable Class Libraries (PCL) and linked files.

Portable Class Libraries Lots of platforms use the .NET Framework. These platforms include Windows, Windows Phone, Xbox, Windows Azure and others supported by Microsoft. When the .NET Framework initially shipped—and up through several iterations afterward—.NET code would need to be recompiled for various platforms. PCLs solve this problem. With a PCL project, you set up a library to support a defined set of APIs available for the target platforms. This choice of platforms is set in the class library's project settings.

Along with Microsoft's support for PCLs, last fall Microsoft changed its PCL licensing to allow support on non-Microsoft platforms. This allowed Xamarin Inc. to provide support for the defined Microsoft PCLs in the iOS and Android platforms and on OS X. Documentation on using PCLs is readily available.

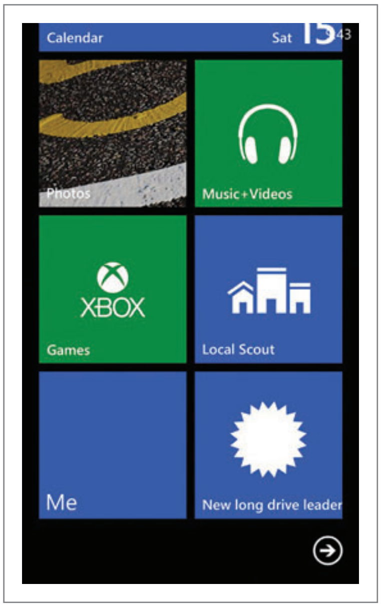


Figure 10 **A Push Message Showing a New Long Drive Leader**

Linked Files PCLs are a great solution for cross-platform development. However, if a feature is included on one platform but not on another, linked files become the alternative way to share code. A linked file setup includes a basic .NET class library, the platform-specific class libraries and the platform application project. The .NET class library has the general code that's shared across platforms.

The platform-specific library contains two types of files. These are the linked files from the generic .NET class library and code that would have common APIs but different platform-specific implementations. The idea is to "Add As Link" a file from the class library project into the platform-specific class library. This is shown in **Figure 8**.

Other Options PCLs and linked files are just two of the options you can use to share code. Other options include partial classes, if/def compiler options, the observer pattern, Xamarin.Mobile (and similar libraries), other libraries available via NuGet (or the Xamarin Component Store) and more.

Sharing code between platforms is an important consideration and you can do this in several ways with the cross-platform capabilities of C# and Xamarin.

Partial classes allow for the multiple class files to be shared across the shared class library and the platform-specific class library. By default, the namespaces will be different between the .NET class library and the platform-specific library. The biggest issue to be aware of with partial classes is that the namespaces must match. Not having the namespaces match is a common mistake made with partial classes.

Visual Studio allows code to be compiled into or out of code via if/then compiler options. Along with this, the platforms can be defined as conditional compilation symbols. These are set up in the project's properties, as shown in **Figure 9**, where the #if directive is used to conditionally compile code for Windows Phone.

Xamarin.Mobile is a set of libraries with common APIs. The libraries are available for Windows Phone, iOS and Android. Xamarin.Mobile currently supports location services, contacts and the camera. I used Xamarin.Mobile geolocation APIs in this app.

To determine location, the geolocator object is a wrapper for the platform-specific geolocation object. Here, it uses the C# 5.0 async-style syntax. Once a location is determined, a call is made into .ContinueWith and processing occurs:

```
geo = new Geolocator();
...
await geo.GetPositionAsync(timeout: 30000).ContinueWith(t =>
  {
    first = t.Result;
    LandingSpot.IsEnabled = true;
  }, TaskScheduler.FromCurrentSynchronizationContext());
```

Note that devices tend to provide geolocation as an approximation. As a result, not every recorded distance will be perfectly accurate.

When apps are built, developers tend to think of higher logical layers of an app calling down into lower levels. For example, a user can touch a button that triggers a location detection. The problem comes when the logical lower level of an app needs to call into a higher layer. A simple solution is to pass a reference from the higher level into a lower level. Unfortunately, this will almost definitely keep the lower-level code from being shared across platforms. This can be overcome with events. Fire an event in the lower level that's processed in a higher level. This is the basis of the observer pattern.

Numerous third parties have created libraries that can be used across platforms. You can find these with NuGet and the Xamarin Component Store.

Push Notifications

Sometimes the server app needs to communicate with the mobile device. This can be done via WAMS or the Notification Hub. WAMS is a great solution when sending a small number of messages. The Notification Hub is designed for sending messages to a large number of devices, such as "premium customers" or "all customers in the state of California." I'll discuss the WAMS option.

You can call WAMS push notifications to a mobile device within the server scripts. Windows Azure handles many of the complex parts of push notifications, but it can't abstract every difference in the way messages are sent to every different platform. Thankfully, the differences are slight.

The mpns object is used to send messages via the Microsoft Push Notification Service (MPNS). The mpns object contains four members. These members are sendFlipTile, sendTile, sendToast and sendRaw. Each of these members has a similar signature. The first parameter is the channel that will be used to communicate. The second parameter is a JSON object with the parameters to be sent to the device. The third parameter is a set of callbacks that occur on the success or failure of the request. The following code using the mpns object is used in the Windows Azure server scripts to send a message if there's a new leader in the longest drive contest:

```
push.mpns.sendFlipTile(item.ChannelUri, {
  title: "New long drive leader"
}, {
    success: function (pushResponse) {
      console.log("Sent push:", pushResponse);
    }
```

Figure 11 Processing a Message on an iPhone

```
public override void RegisteredForRemoteNotifications(
  UIApplication application, NSData deviceToken)
{
  string trimmedDeviceToken = deviceToken.Description;
  if (!string.IsNullOrWhiteSpace(trimmedDeviceToken))
  {
    trimmedDeviceToken = trimmedDeviceToken.Trim('<');
    trimmedDeviceToken = trimmedDeviceToken.Trim('>');
  }
  DeviceToken = trimmedDeviceToken;
}

public override void ReceivedRemoteNotification(
  UIApplication application, NSDictionary userInfo)
{
  System.Diagnostics.Debug.WriteLine(userInfo.ToString());
  NSObject inAppMessage;

  bool success = userInfo.TryGetValue(
    new NSString("inAppMessage"), out inAppMessage);

  if (success)
  {
    var alert = new UIAlertView("Got push notification",
      inAppMessage.ToString(), null, "OK", null);
    alert.Show();
  }
}
```

The result is the update to the tile as shown in **Figure 10**. Notice in the image how the tile has been updated to say "New long drive leader."

You use an apns object to send messages to the Apple Push Notification Services (APNS) in the WAMS scripts. It's conceptually similar to the mpns object. The member of most interest is the send method. It has a signature similar to the mpns send methods. The signature contains three parameters: the deviceToken, which uniquely identifies a device; a JSON-based parameter object; and a final parameter that's a set of callbacks.

> Partial classes allow for the multiple class files to be shared across the shared class library and the platform-specific class library.

Here's the code showing how the apns object is used to send a "new leader" message to an iOS device:

```
push.apns.send(item.deviceToken, {
  alert: "New Long Drive",
  payload: {
    inAppMessage: "Hey, there is now a new long drive."
  }
});
```

Figure 11 shows the code to be added to the AppDelegate.cs file to handle the message sent to the iPhone. In this example, a UIAlertView is displayed to the user.

If needed, you can use a gcm object to send messages to the Google Cloud Messaging (GCM) platform.

One significant difference between the Windows and Apple push notifications (and Google notifications) is how the client mobile system handles these messages. A full listing of client systems is included in the Xamarin.iOS project in the AppDelegate.cs file in the accompanying code download.

And that's it. Good luck with your mobile app development and your golf game! ■

Wallace B. McClure *graduated from the Georgia Institute of Technology (Georgia Tech) with bachelor's and master's degrees in electrical engineering. He has done consulting and development for companies large and small. McClure has authored books on iPhone programming with Xamarin.iOS; Android programming with Xamarin.Android; application architecture; ADO.NET and SQL Server; and AJAX. He is a Microsoft MVP, ASPInsider, Xamarin MVP and Xamarin Insider. He's also a partner in Scalable Development Inc. His training materials for iOS and Android are available via Learn Now Online. His blog is at morewally.com and he's on Twitter at twitter.com/wbm.*

Thanks to the following technical experts for reviewing this article:
Kevin Darty (independent contractor) and Brian Prince (Microsoft)

Build an Advanced Camera App for Nokia Lumia Phones

Rajesh Lal

In this article, I'm going to teach you how to develop an app for the 41-megapixel (MP) Nokia Lumia 1020 and 20MP Nokia Lumia 1520 smartphones. I'll focus primarily on the Nokia Lumia 1020, but the information applies to all Lumia Windows Phone 8 devices with PureView technology. First, I'll discuss PureView, the technology behind the powerful camera included in the phones, and then I'll explain the advanced features available to access and enhance your photographs. I'll provide an overview of the Nokia Imaging SDK (it has tons of ready-to-use graphic filters) and a walk-through of how to use it. I'll also cover the typical workflow needed to build a camera app and show you how to create a tilt-shift photo filter to simulate shallow depth of field. Let's get started.

Understanding the PureView Technology in a 41MP Camera Phone

PureView technology consists of advanced camera hardware and related software. Together, they allow the capturing and saving of high-quality, high-resolution images. The three main aspects of PureView technology are a high-resolution camera lens, oversampling and lossless zoom. I'll briefly explain each one.

The core of PureView technology is a high-resolution sensor, 7,728 pixels wide and 5,368 pixels high, totaling more than 41MP. This enables the camera phone to capture big photographs, six times larger than a normal 5MP photograph.

Figure 1 compares a 41MP resolution picture with a 5MP resolution picture. Because of this 41MP resolution, you can take high-quality 34MP 16:9 photographs (7,728 x 4,354) as well as 38MP 4:3 photographs (7,152 x 5,368), as shown in the lens view of the camera in **Figure 2**.

Pixel oversampling is when the camera takes a 41MP photograph and creates a high-quality 5MP image. This is the picture you see on the phone's screen. Furthermore, the oversampling process keeps all the rich detail in the image but filters away any visual noise.

The third aspect of PureView technology is lossless zoom, which simply means you don't lose image quality when you zoom. This

This article discusses:

- PureView technology used in advanced Nokia smartphones
- Random Access JPEG technology
- Nokia Imaging SDK APIs
- Camera app workflow
- Creating a tilt-shift photographic filter

Technologies discussed:

Windows Phone 8, Nokia Imaging SDK

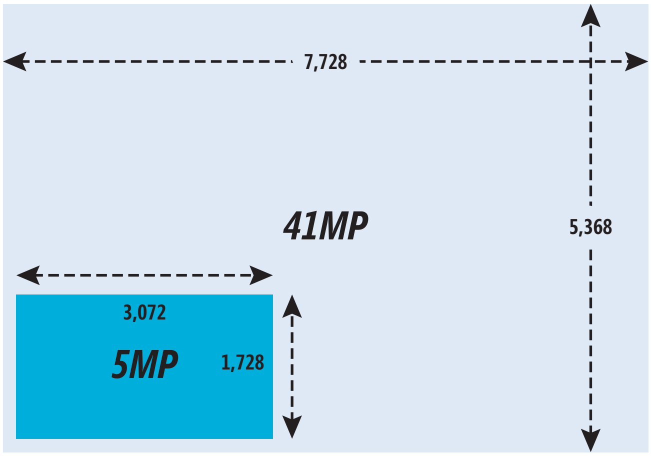


Figure 1 **A 41MP Picture Compared with a 5MP Picture**

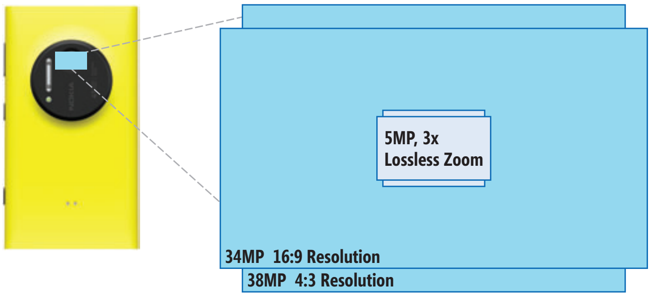


Figure 2 **Lossless Zoom Is Still Part of the Photograph**

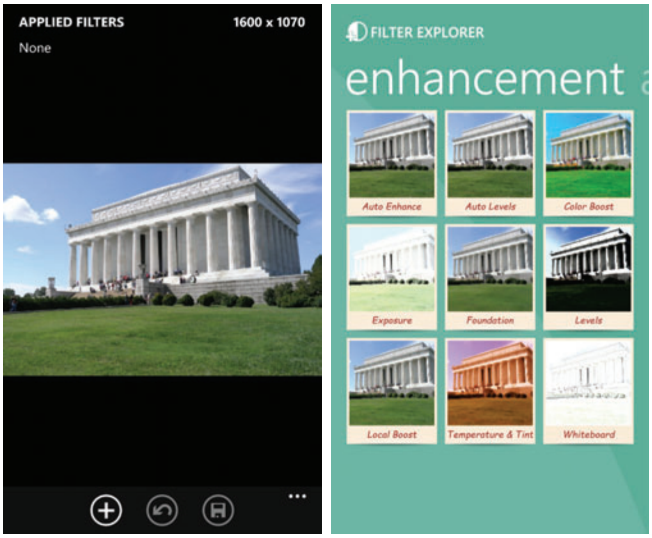


Figure 3 **Enhancement Filters in the Free App Filter Explorer**

is possible because—at any point in time—you have available the original 41MP picture, which is oversampled to show a 5MP photo. *When you zoom, you're really zooming into a part of the original 41MP photo*. At up to a 3x zoom, you're still dealing with parts of the original 41MP photo. And even at maximum zoom, your photo quality is still the quality of a regular 5MP photo. Due to oversampling, the picture quality only gets better as you zoom. **Figure 2** shows how the lossless zoom is still part of the original photograph.

The Nokia Imaging SDK

When building a camera app, a high-resolution photograph is the raw material, but you'll need an advanced software stack that can use the huge images and let you access and manipulate them. This is where the Nokia Imaging SDK comes into the picture (bit.ly/1hJkmpl). It provides a set of advanced features to access a high-resolution photograph taken by the camera.

The Nokia Imaging SDK includes partial JPEG decoding, called Random Access JPEG (RAJPEG) technology. RAJPEG allows for random access of JPEG data, fast downscaling of images and instant partial decoding. These aspects of RAJPEG enable real-time image manipulation. The Nokia Imaging SDK contains more than 50 filters, effects and enhancements. It even contains a set of the most common image operations, such as crop, resize, rotate and undo, just to name a few. These out-of-the-box features help you create advanced camera/photo-centric applications without worrying about the most common functionalities. To use the Nokia Imaging SDK in your project, follow the instructions on the Nokia Lumia Developer Library page (bit.ly/KzDPNG).

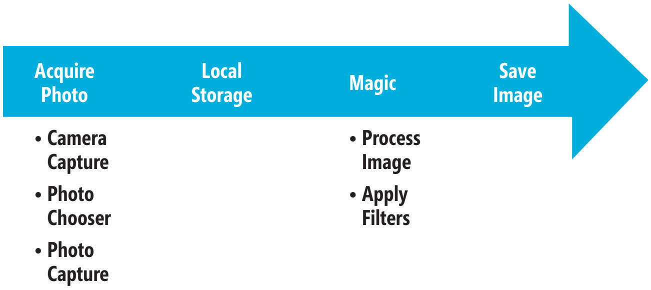


Figure 4 **Image App Workflow**

Imaging SDK APIs

The APIs available in the Imaging SDK can be easily applied to the image data captured by the camera in relatively few lines:

```
var imageSource = new StreamImageSource(e.ChosenPhoto);
// Define the effects to apply
var filterEffect = new FilterEffect(imageSource);
filterEffect.Filters = new [] { new FogFilter() };

// Render the image using WriteableBitmapRenderer
WriteableBitmapRenderer renderer =
  new WriteableBitmapRenderer(filterEffect,
  myImageBitmap,OutputOption.PreserveAspectRatio);
await renderer.RenderAsync();
```

Here, e.ChosenPhoto is used to create a StreamImageSource. The e.ChosenPhoto can come directly from a camera-captured photo or from any photo in the album. I then created a filter effect to apply on the photo and added an artistic FogFilter to the array of filters in my effect. If desired, this array can contain multiple filters, which are then applied and rendered to myImageBitmap using a WriteableBitmapRenderer. As the name suggests, it renders an image source to a WriteableBitmap. Finally, the RenderAsync method helps in the asynchronous rendering of the image.

Along with FogFilter, you can apply more than 50 other Imaging SDK filters to a photograph. **Figure 3** shows the list of enhancement

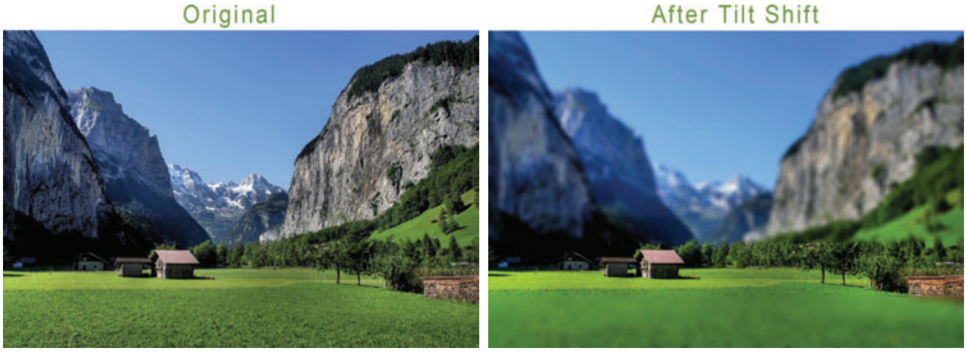


Figure 5 **The Tilt-Shift Effect**

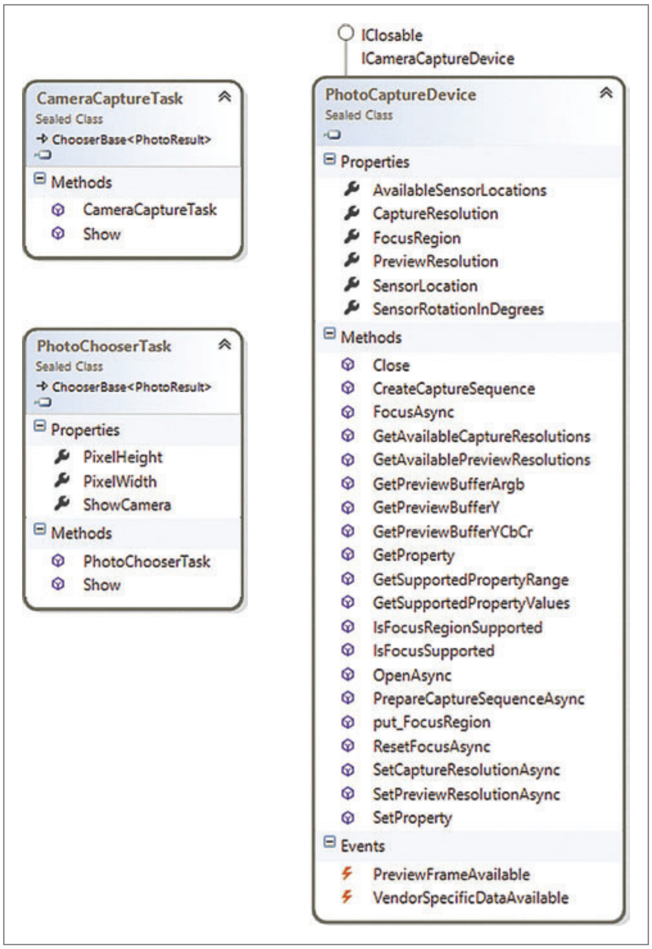


Figure 6 **Methods and Properties Supported by PhotoCaptureDevice**

filters inside the free App Filter Explorer, available out of the box for developers. It includes filters such as AutoEnhanceFilter, AutoLevelsFilter, ColorBoostFilter and ExposureFilter. These are filters you can directly apply to any photo or real-time captured image to enhance quality.

There are other filters for brightness (BrightnessFilter) and contrast (ContrastFilter), which take parameters for brightness and contrast. There are a number of filters just for artistic effects, such as emboss, grayscale and sepia. There are also the image-editing “filters” that provide the previously mentioned commonly used functions such as rotate, flip, mirror and so on. These can be applied to a single image multiple times, one on top of another, to create dramatic effects.

High-Resolution Camera App Workflow

Now that I’ve told you about the hardware and Imaging SDK capabilities, I’ll show you the typical workflow you need for building a powerful camera phone app. Because I’m dealing with a huge image, I have to play nice so my app doesn’t make other apps slow or take a huge amount of memory. This can be ensured by not trying to directly manipulate the big image in the camera.

At any point of time, the camera will have two photographs: the original 41MP photograph and the 5MP photograph, which is shown in the camera view panel. In my app, I’ll always work on the 5MP camera phone picture, which can be either the oversampled image of the whole photo, the zoomed part of the picture, or the oversampled and partially zoomed part of the photo.

Figure 7 **The XAML UI Code**

```
<Canvas x:Name="LayoutRoot" Background="Transparent">
  <TextBlock Text="" Style="{StaticResource
    PhoneTextNormalStyle}" />
  <Image x:Name="TiltShiftImage" Height="480" Width="728"
    Stretch="UniformToFill" MouseLeftButtonUp
    ="TiltShiftImage_MouseLeftButtonUp"/>
  <Image x:Name="OriginalImage" Height="480" Width="728"
    Stretch="UniformToFill" Canvas.ZIndex="0"
    MouseLeftButtonUp="OriginalImage_MouseLeftButtonUp"
    Source="/Assets/Landscapes.jpg"/>
  <Rectangle x:Name="TiltshiftRegion" Fill="White" Height="65"
    Stroke="#FF0B7AFF" Canvas.Top="320" Width="728"
    Opacity="0.25" StrokeThickness="5"/>
  <Button x:Name="SelectButton" Content="Select"
    Click="PickAnImageButton_Click" Canvas.Left="4"
    Canvas.Top="398" />
  <Button x:Name="CameraButton" Content="Camera"
    Click="CameraButton_Click" Canvas.Left="123"
    Canvas.Top="398" />
  <Button x:Name="ProButton" Content="Pro Camera"
    Click="ProCameraButton_Click" Canvas.Left="254"
    Canvas.Top="398" />
  <Button x:Name="SaveButton" Content="Save"
    Click="SaveImage_Click" Canvas.Left="630"
    Canvas.Top="398" />
  <Button x:Name="TiltShiftButton" Content="Tilt Shift"
    Click="TiltShiftButton_Click" Canvas.Left="449"
    Canvas.Top="398" />
</Canvas>
```

Figure 4 shows the workflow for the imaging app.

The individual steps include:

- Capture a high-resolution image.
- Save it to your local storage.
- Do your magic using the Imaging SDK.
- Scale down the resulting image to 5MP.
- Save the 5MP-enhanced image to the camera roll.

Creating a Photographic Filter: Tilt Shift

Now I’ll walk through the workflow of my photographic filter-based, tilt-shift camera app. Tilt-shift photography is the use of camera movements—specifically tilt—for selective focus to simulate a miniature scene (see **Figure 5**). You can find more information on tilt-shift photography at bit.ly/1bRYYNK. In this app, I’ll simulate the shallow depth of field with digital post-processing using multiple filters in the Nokia Imaging SDK.

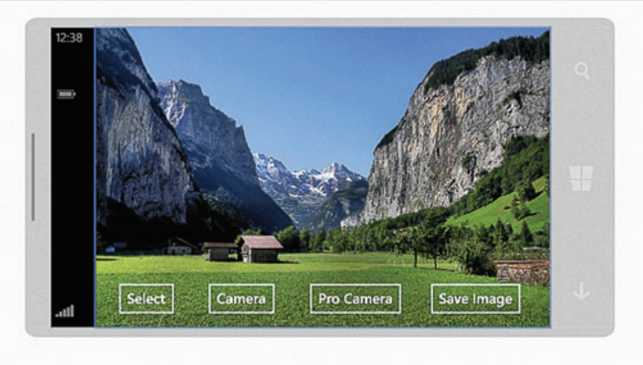


Figure 8 **The Tilt-Shift App UI**

SEE THE WORLD AS A DATABASE
ADO.NET ▪ JDBC ▪ ODBC ▪ SQL SSIS ▪ ODATA
MYSQL ▪ EXCEL ▪ POWERSHELL
Visual Studio
Java
ODBC
SQL Server
Excel
BizTalk
MySQL
OData

Figure 9 **High-Resolution Options**

Models Variants	Manually Configurable	High-Resolution Options
Lumia 1020	RM-875, RM-876, RM-877	7712 x 4352 (16:9), 7136 x 5360 (4:3)
Lumia 1520	RM-937, RM-938, RM-939	5376 x 3024 (16:9), 4992 x 3744 (4:3)

I'll show you how this fits in the workflow I discussed earlier.

Acquire Photo The first step for any camera-based app is to acquire an image either from the phone's camera or image gallery. You can do this in three different ways. One is using the default 5MP camera by using the CameraCaptureTask, which triggers the normal camera viewfinder. The second method is by using the native photo picker control called by PhotoChooserTask. These two alternatives are sufficient for most camera apps, but for an advanced app where you need to capture high-resolution photos, you need to create a custom viewfinder, which triggers the built-in Pro Camera application viewfinder with high resolution. This is done using the object PhotoCaptureDevice. See **Figure 6** for the list of methods and properties supported by PhotoCaptureDevice.

The UI The tilt-shift app consists of two images: one OriginalImage to show the actual image captured or selected using the photo chooser, and the TiltShiftImage, which displays the final image after the tilt-shift filter is applied to the original image. The SelectButton triggers the native photo chooser, and the CameraButton triggers the camera, as shown in the code in **Figure 7**.

The resulting camera UI is shown in **Figure 8**.

Figure 10 **Initializing the Pro Camera Application**

```
private void InitializeCamera() {
  Windows.Foundation.Size captureResolution;
  var deviceName = DeviceStatus.DeviceName;
  if (deviceName.Contains("RM-875") || deviceName.Contains("RM-876") ||
    deviceName.Contains("RM-877"))
  {
    captureResolution = new Windows.Foundation.Size(7712, 4352); // 16:9
    //captureResolution = new Windows.Foundation.Size(7136, 5360); // 4:3
  }
  else if (deviceName.Contains("RM-937") || deviceName.Contains("RM-938") ||
    deviceName.Contains("RM-939"))  {
      captureResolution = new Windows.Foundation.Size(5376, 3024); // 16:9
      //captureResolution = new Windows.Foundation.Size(4992, 3744); // 4:3
    }
  else {
    captureResolution = PhotoCaptureDevice.GetAvailableCaptureResolutions(
      REAR_CAMERA_SENSOR_LOCATION).First();
  }
  var task = PhotoCaptureDevice.OpenAsync(REAR_CAMERA_SENSOR_LOCATION,
  captureResolution).AsTask();
  task.Wait();
  _device = task.Result;
  _device.SetProperty(
    KnownCameraGeneralProperties.PlayShutterSoundOnCapture, true);
  if (_flashButton != null) {
    SetFlashState(_flashState);
  }
  AdaptToOrientation();
  ViewfinderVideoBrush.SetSource(_device);
  if (PhotoCaptureDevice.IsFocusSupported(REAR_CAMERA_SENSOR_LOCATION))  {
    Microsoft.Devices.CameraButtons.ShutterKeyHalfPressed +=
    CameraButtons_ShutterKeyHalfPressed;
  }
  Microsoft.Devices.CameraButtons.ShutterKeyPressed +=
    CameraButtons_ShutterKeyPressed;
}
```

Figure 11 **Capturing an Image via Shutter Key-Pressed Events**

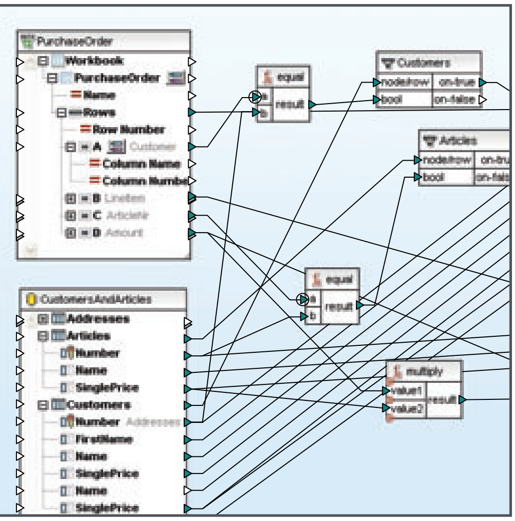
```
private async void CameraButtons_ShutterKeyHalfPressed(
  object sender, EventArgs e) {
    if (!_focusing && !_capturing) {
      _focusing = true;
      await _device.FocusAsync();
      _focusing = false;
    }
}

private async void CameraButtons_ShutterKeyPressed(
  object sender, EventArgs e) {
    if (!_focusing && !_capturing) {
      _capturing = true;
      var stream = new MemoryStream();
      try {
        var sequence = _device.CreateCaptureSequence(1);
        sequence.Frames[0].CaptureStream = stream.AsOutputStream();
        await _device.PrepareCaptureSequenceAsync(sequence);
        await sequence.StartCaptureAsync();
      }
      catch (Exception ex) {
        stream.Close();
      }
      _capturing = false;
      if (stream.CanRead) {
        // Process the image in the stream
        // This can be saved to the local storage
      }
    }
}
```

Capture a High-Resolution Image The first two methods of normal camera and photo picker functionality are straightforward. The Microsoft.Phone.Tasks namespace has two task objects, CameraCaptureTask and PhotoChooserTask, for these two purposes. Simply select the image either from the photo chooser or from the result of the camera capture as the source of your tilt-shift filter:

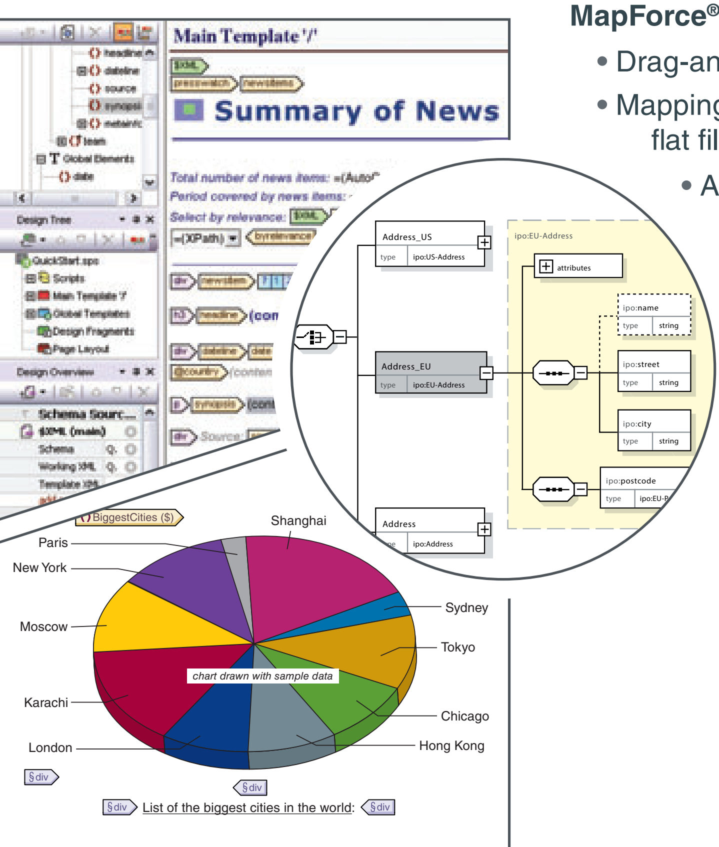
```
private void PickAnImageButton_Click(object sender, RoutedEventArgs e)
{
  PhotoChooserTask chooser = new PhotoChooserTask();
  chooser.Completed += PickImageCallback;
  chooser.Show();
}
private void CameraButton_Click(object sender, RoutedEventArgs e)
{
  CameraCaptureTask camera = new CameraCaptureTask();
  camera.Show();
  camera.Completed += PickImageCallback;
}
```

Figure 12 **Initializing Filters for an Effect**

```
public partial class MainPage : PhoneApplicationPage {
  private Stream _img;
  private StreamImageSource imageSource;
  private ImageProviderInfo info;
  private FilterEffect _tiltshiftEffect = null;
  private WriteableBitmap _tiltshiftImageBitmap = null;
  // Constructor
  public MainPage()  {
    InitializeComponent();
    _tiltshiftImageBitmap =
    new WriteableBitmap((int)TiltShiftImage.Width,(int)TiltShiftImage.Height);
  }
...
private async void PickImageCallback(Object sender, PhotoResult e) {
  if (e.TaskResult != TaskResult.OK) {
    return;
  }
  _img = e.ChosenPhoto;
  imageSource = new StreamImageSource(_img);
  var bitmapImage = new BitmapImage();
  bitmapImage.SetSource(e.ChosenPhoto);
  OriginalImage.Source = bitmapImage;
  info = await imageSource.GetInfoAsync();
  TiltShift();
...
```



PurchaseOrder
Workbook
PurchaseOrder
Name
Rows
Row Number
Customer
Column Name
Column Number
Lineitem
Articlenr
Amount
equal
result
Customers
node/row
on-true
bool
on-false
Articles
CustomersAndArticles
Addresses
Articles
Number
Name
SinglePrice
Customers
FirstName
multiply
value1
value2



Main Template '/'
Summary of News
Total number of news items:
Period covered by news items:
Select by relevance:
Design Tree
Design Overview
Schema Source
Address_US
ipo:US-Address
Address_EU
ipo:EU-Address
Address
ipo:Address
attributes
ipo:name
ipo:street
ipo:city
ipo:postcode
string
BiggestCities ($)
Shanghai
Paris
New York
Moscow
Karachi
London
Sydney
Tokyo
Chicago
Hong Kong
chart drawn with sample data
List of the biggest cities in the world:

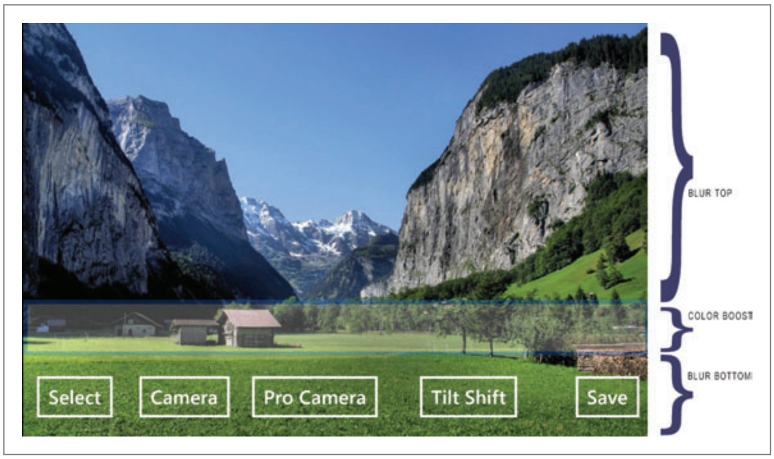


Figure 13 **The Three Rectangles Used for the Tilt-Shift Effect**

To capture a high-resolution photo, I need to create a custom viewfinder using a video brush whose source will be the image of the Pro Camera application:

```
<Canvas x:Name="Canvas" Tap="Canvas_Tap" Height="480"
  HorizontalAlignment="Center" VerticalAlignment="Center">
  <Canvas.Background>
    <VideoBrush x:Name="ViewfinderVideoBrush" Stretch="Uniform"/>
  </Canvas.Background>
  <Border x:Name="FocusBracket" Width="80" Height="80"
    BorderBrush="White" BorderThickness="2" Margin="-40"
    Visibility="Collapsed" CornerRadius="360"/>
  <Image x:Name="FreezeImage" Visibility="Collapsed"
    Stretch="Uniform" Height="480"/>
</Canvas>
```

Figure 14 **Combining Filters for an Effect**

```
private async void TiltShift() {
  if (info == null) return;
  try {
    var topLeft = new Windows.Foundation.Point(0, 0);
    var tiltShiftRegionTop = Canvas.GetTop(TiltshiftRegion);
    var delta = info.ImageSize.Height / TiltShiftImage.Height;
    tiltShiftRegionTop = (tiltShiftRegionTop + 10) * delta;
    var tiltShiftRegionBottom =
      (Canvas.GetTop(TiltshiftRegion) + 10) * delta + TiltshiftRegion.Height;

    var topRight =
      new Windows.Foundation.Point(info.ImageSize.Width, tiltShiftRegionTop);
    var bottomLeft = new Windows.Foundation.Point(0, tiltShiftRegionBottom);
    var bottomRight = new Windows.Foundation.Point(
      info.ImageSize.Width, info.ImageSize.Height);
    // Define the effects to apply
    _tiltshiftEffect = new FilterEffect(imageSource);

    List<IFilter> filters = new List<IFilter>();
    filters.Add(new BlurFilter(15, (
      new Windows.Foundation.Rect(topLeft, topRight)),
      BlurRegionShape.Rectangular));
    filters.Add(new ColorBoostFilter(0.5));
    filters.Add(new BlurFilter(23, (new Windows.Foundation.Rect(bottomLeft,
      bottomRight)), BlurRegionShape.Rectangular));

    _tiltshiftEffect.Filters = filters;
    // Render the image using WriteableBitmapRenderer
    WriteableBitmapRenderer renderer =
      new WriteableBitmapRenderer(_tiltshiftEffect,
      _tiltshiftImageBitmap, OutputOption.PreserveAspectRatio);
    _tiltshiftImageBitmap = await renderer.RenderAsync();
    TiltShiftImage.Source = _tiltshiftImageBitmap;
  }
  catch (Exception exception) {
    MessageBox.Show("Exception:" + exception.Message);
    return;
  }
  SaveButton.IsEnabled = true;
}
```

I also need to initialize the Pro Camera application with the correct resolution depending on the device, Lumia 1020 or Lumia 1520, and the type of resolution wanted. See **Figure 9** for options.

Figure 10 shows how you can initialize the Pro Camera application.

Capturing the image from the Pro Camera application involves ShutterHalfKeyPressed and ShutterKeyPressed events, as shown in **Figure 11**.

Do the Magic First I'll use the PickImageCallback to get a photo, which is then set as a source of the OriginalImage. To access dimensions of the image, I'll use ImageProviderInfo.

I'll also create filters and apply them in the session. To create a tilt-shift effect, I'll use three different filters: BlurFilter with KernelSize 15; ColorBoostFilter with value 0.5; and again BlurFilter with KernelSize 21. The two blur filters make the foreground and the background of the image out of focus and the ColorBoostFilter brightens the region, which I want to do to create the miniature effect. The code for this is shown in **Figure 12**.

To apply the tilt-shift filter, I need three rectangles: the top rectangle for blur; the middle rectangle for color boost; and the bottom rectangle for blur again (see **Figure 13**). In this example, I'll use a rectangular region called TiltshiftRegion, which the user can touch and move to customize the position where the tilt shift takes place. The translucent rectangle shown in **Figure 13** becomes tilt shifted and the rest of the area is blurred.

The TiltShiftRegion position is used to calculate the three rectangles where the filters are applied (see **Figure 14**).

Save Photo Finally, the processed image needs to be saved back, as shown in **Figure 15**.

And I'm done. You now understand PureView advanced camera capture and post processing of the image using the Nokia Imaging SDK. I hope you found this article useful, and I look forward to hearing your comments and suggestions. ■

RAJESH LAL *works at Nokia and is passionate about Windows Phone and Web technologies. His latest book is "Digital Design Essentials: 100 Ways to Create Better Desktop, Web and Mobile Interfaces" (Rockport Publishers, 2013). You can find it at bit.ly/dsgnbk. For information about the author, check out iRajLal.com.*

THANKS to the following technical experts for reviewing this article: *Chevon Christie (Microsoft) and Paras Wadehra (Nokia)*

Figure 15 **Saving the Processed Image**

```
private async void SaveImage_Click(object sender, RoutedEventArgs e) {
  SaveButton.IsEnabled = false;
  if (_tiltshiftEffect == null) {
    return;
  }
  var jpegRenderer = new JpegRenderer(_tiltshiftEffect);
  // JPEG renderer gives the raw buffer for the filtered image
  IBuffer jpegOutput = await jpegRenderer.RenderAsync();

  // Save the image as a JPEG to the saved pictures album
  MediaLibrary library = new MediaLibrary();
  string fileName = string.Format("TiltShiftImage_{0:G}", DateTime.Now);
  var picture = library.SavePicture(fileName, jpegOutput.AsStream());
  MessageBox.Show("Tilt Shift Image saved!");
  SaveButton.IsEnabled = true;
}
```

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

REDMOND 2014

SET YOUR COURSE FOR 127.0.0.1!

vslive.com/redmond

PLATINUM SPONSOR esri

SPONSOR LogiGear Software Testing

SUPPORTED BY Microsoft | Visual Studio | msdn magazine | Visual Studio MAGAZINE

YOUR GUIDE TO THE .NET DEVELOPMENT UNIVERSE

From August 18 – 22, 2014, developers, engineers, software architects and designers will land in Redmond, WA at the idyllic Microsoft headquarters for 5 days of cutting-edge education on the Microsoft Platform.

Come experience code at the source – rub elbows with Microsoft stars, get the inside scoop on what's next, and learn what you need to know now. Over 5 days and 60+sessions and workshops, you'll explore the .NET Development Universe, receiving the kind of practical, unbiased training you can only get at Visual Studio Live!

Tracks Include:

- Visual Studio/.NET Framework
- Windows Client
- JavaScript/HTML5 Client
- ASP.NET
- Cloud Computing
- Windows Phone
- Cross-Platform Mobile Development
- SharePoint
- SQL Server

PRODUCED BY

1105 MEDIA INC

REDMOND 2014

August 18 – 22
Microsoft Campus
Redmond, WA

Register by June 11 and Save $400!

Use promo code VSLAPR2

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

CONNECT WITH VISUAL STUDIO LIVE!

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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

vslive.com/redmond

Using Survival Analysis

Zvi Topol

Survival analysis (SA) is a discipline of statistics that focuses on estimating time to events. You would typically apply survival analysis methods to clinical studies to help determine the effectiveness of certain drugs (time to patient death), reliability of software systems (time to failure) and credit analytics (time to loan default).

Pharmaceutical clinical studies involving two groups of patients are an excellent example of how this can work. The control group members are administered a placebo. The test group members receive the experimental drug targeting the disease. Survival analysis methods are then applied to determine whether there's a statistically significant difference in patient survival between the two groups. The "time to" event in this case is the time from the beginning of the study until patients die.

To expose you to using SA, I'll cover basic concepts along with a C# implementation of a commonly used estimation method called the Kaplan-Meier estimator. I'll use a real-world example of estimating the survival probability of mobile applications.

This article discusses:

- How to use survival analysis to estimate time to events
- Using a C# implementation of the Kaplan-Meier estimating algorithm
- Using censoring to account for missing data
- Coding the Kaplan-Meier algorithm and interpreting the results

Technologies discussed:

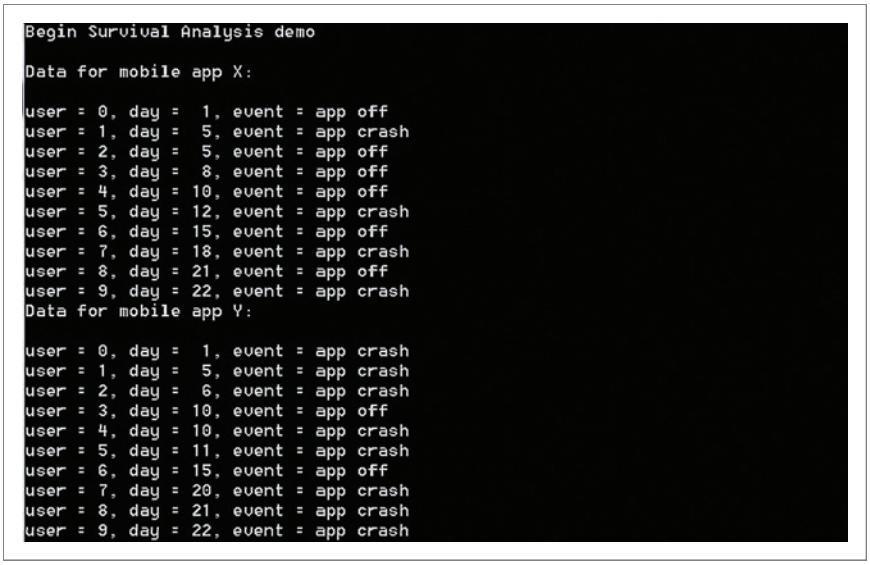
C#

Imagine a software development firm produces two separate mobile applications titled X and Y. Each one is developed by separate teams. The firm is eager to learn how robust the mobile applications are and to determine if one application is significantly less robust and requires more effort to improve its reliability.

At any given moment, there can be many instances of X and Y alive and running on customer mobile devices. Thus, a mobile application crash is what's most interesting. Longer time periods to the event, in this case, will indicate the application is more robust or has better survivability.

The most basic concept in SA is that of the survival function.

In the demo program, I'll first display the survival data for mobile applications X and Y (see **Figure 1**). The data shows both applications were run by 10 different users with IDs ranging from zero to nine. In my example, an application can either crash (described by event = app crash in the screenshot) or get closed by the user (described by event = app off). The day on which the event occurs is also recorded.



```
Begin Survival Analysis demo

Data for mobile app X:

user = 0, day =  1, event = app off
user = 1, day =  5, event = app crash
user = 2, day =  5, event = app off
user = 3, day =  8, event = app off
user = 4, day = 10, event = app off
user = 5, day = 12, event = app crash
user = 6, day = 15, event = app off
user = 7, day = 18, event = app crash
user = 8, day = 21, event = app off
user = 9, day = 22, event = app crash
Data for mobile app Y:

user = 0, day =  1, event = app crash
user = 1, day =  5, event = app crash
user = 2, day =  6, event = app crash
user = 3, day = 10, event = app off
user = 4, day = 10, event = app crash
user = 5, day = 11, event = app crash
user = 6, day = 15, event = app off
user = 7, day = 20, event = app crash
user = 8, day = 21, event = app crash
user = 9, day = 22, event = app crash
```

Figure 1 **The Survival Analysis Demo Showing Lifecycle of Mobile Apps**

Basic Concepts of SA

The most basic concept in SA is that of the survival function. This is commonly denoted by S(t). If X is a random variable (a variable whose values are outcomes based on chance) representing the event time, then S(t) = Pr (X > t). In other words, S(t) is the probability for survival after time t. S(0) is defined to be 1. The survival function is related to the lifetime distribution function. This is typically denoted by F(t) and is defined as F(t) = Pr(X<=t), in other words, the probability the event happened until time t. Therefore, F(t) = 1 – S(t). The event density function f(t) is then defined to be dF(t)/dt—the first derivative of F(t), if F(t) is differentiable. Therefore, f(t) can be thought of as the rate of the event (per unit of time).

The hazard function, another basic concept, is equal to f(t)/S(t) and is the event rate at time t for individuals who are alive at time t.

You can specify survival functions parametrically, using an explicit function or a family of functions. You can also infer them non-parametrically from existing data, without having a parametrized closed form. A semi-parametric specification, which is a mix between parametric and non-parametric specification, is also possible. The exponential distribution is a simple and popular parametrized function family to describe survival functions due to its appealing mathematical properties.

For example, S(t) = exp(-0.05t) is a survival function from a paramterized exponential distribution plotted in **Figure 2**. The survival functions of the form S(t) = exp(-at) (where a is a parameter controlling the hazard rate can describe that distribution). The lifetime distribution function is given by F(t) = 1 – S(t) = 1 – exp(-at). **Figure 2** helps us visualize how survival functions behave over time.

Working with a given parametric model, you can use actual data to estimate the model's parameters. In the case of exponential distribution, it's the parameter a. One way to do so is to use Maximum Likelihood Estimation (MLE) methods, but that's another subject entirely.

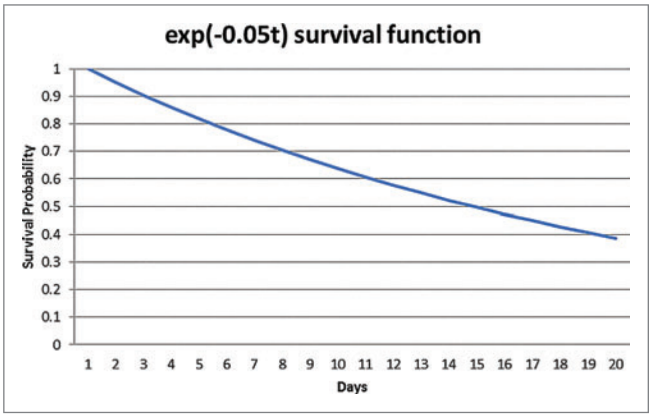


Figure 2 **How Survival Functions Behave over Time**

I'll focus on implementing a non-parametric estimate for the survival function. That is, I won't set a predefined model for the survival function and estimate the model parameters. Instead, I'll derive the survival function directly from the data observed. Before I describe how to do that, I have to explain another important concept of SA called censoring.

Censoring occurs when some observations in the dataset are incomplete. At some point, you've lost track of the item observed. In my example, this would mean a mobile application ended its execution without crashing (throwing a fatal exception). The application was gracefully closed by the user. Although there can be other reasons an application ended without crashing, I'll assume an application either crashes or gets closed by the user.

The Kaplan-Meier (KM) estimator is a non-parametric algorithm that estimates the survival function.

There are two main flavors for censoring—right censoring and left censoring. Right censoring occurs when the start time is known, but the event time is missing. Left censoring occurs when the event time is present, but the start time is missing. Right censoring is occuring in my example.

Using the Kaplan-Meier Estimator to Estimate the Survival Function

The Kaplan-Meier (KM) estimator is a non-parametric algorithm that estimates the survival function. Deriving the KM estimator entails the use of advanced math, including martingale theory and counting processes, and is beyond the scope of this article. Implementing the KM estimator, however, is straightforward and is based on counts.

Figure 3 **Survival Data of Mobile Application X**

UserID	Days	Crashed	Censored
0	1		X
1	5	X	
2	5		X
3	8		X
4	10		X
5	12	X	
6	15		X
7	18	X	
8	21		X
9	22	X	

Consider computing the KM estimator for the survival of mobile application X. The KM estimator needs to keep track of three different counts:

1. How many instances of mobile application X are still up and running. This is represented using the variable atRisk in my implementation.
2. The number of instances that have crashed. This is tracked in the crashed variable.
3. The number of instances that finished execution gracefully. These are counted using the variable censored.

The following lines of code (for mobile application X) are using the CrashMetaData class to encode the survival data represented in **Figure 3**:

```
var appX = new CrashMetaData[] {new CrashMetaData{UserID = 0,
  CrashTime = 1, Crashed = false},
            new CrashMetaData{UserID = 1, CrashTime = 5, Crashed = true},
            new CrashMetaData{UserID = 2, CrashTime = 5, Crashed = false},
            new CrashMetaData{UserID = 3, CrashTime = 8, Crashed = false},
            new CrashMetaData{UserID = 4, CrashTime = 10, Crashed = false},
            new CrashMetaData{UserID = 5, CrashTime = 12, Crashed = true},
            new CrashMetaData{UserID = 6, CrashTime = 15, Crashed = false},
            new CrashMetaData{UserID = 7, CrashTime = 18, Crashed = true},
            new CrashMetaData{UserID = 8, CrashTime = 21, Crashed = false},
            new CrashMetaData{UserID = 9, CrashTime = 22, Crashed = true}};
```

The survival data contains event time in days (encoded by CrashTime) and information about whether the event refers to an application crash or censoring. If Crashed is equal to true, the application crashed. Otherwise, the application closed gracefully (in other words, was censored). Additionally, a UserID field is tracking the instance of the application.

The KM estimator is implemented in the EstimateKaplanMeier method. This partitions the data to different non-overlapping time intervals based on time periods to events (in my example this is an application crash). It keeps track of the counts in each interval.

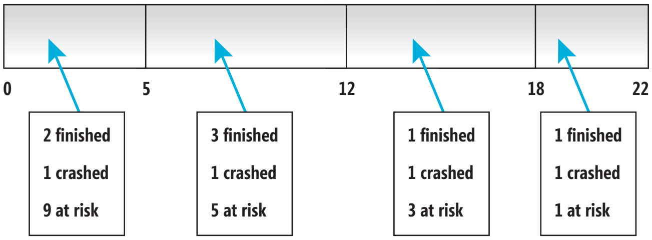


Figure 4 **Day Intervals Created by the KM Estimator**

It's important to note the count of how many applications are still up and running is done just before the event (this is due to the mathematical formulation of counting processes). So in the first interval in my example, which covers days 0 to 5, 9 out of the 10 instances were up and running just before day 5 (one instance finished running at time 1). In the interval up to and including day 5, I had one crash (which defines the interval) and 2 instances finishing (on days 1 and 5). See **Figure 4**.

The KM estimate for the survival function is then the product over all the different intervals of the survival derived from the counts in the partitions:

> 1 – (crashed in interval) /(those at risk just before the end of the interval)

The EstimateKapalanMeier method returns an object of class SurvivalCurve. This represents the estimated survival function. The output is a step function. Each step is the value of survival function in a corresponding interval (as estimated by the KM estimator). **Figure 5** includes part of the Survival Analysis demo program output corresponding to the SurvivalCurve object (for both applications X and Y).

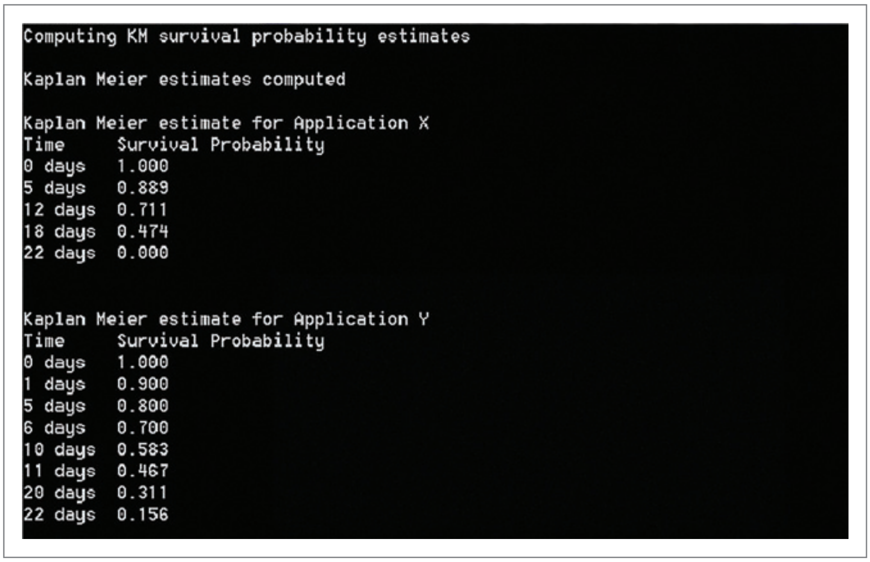
> Another question you can answer using the KM estimates is whether there's a difference in the survivability of two (or more) different applications.

Figure 6 includes a plot of the step survival function estimated for mobile application X. In the plot, short vertical lines in each step denote multiple occurrences of the crash event during the interval corresponding to the step.

You can then use the estimate to infer the median survival time, or the time by which half of the instance will be alive. This should occur at some point in time between days 12 (where the survival probability estimate is 0.711 > 0.5) and 18 (where the survival probability is 0.474 < 0.5). There are a few approaches in the SA literature describing how to exactly compute this quantity, because it typically falls between two steps.

I'll define the median survival time as the minimal survival time for which the survival function is less than 0.5, which for mobile application X results in a median survival time of 18 days. The interpretation of this quantity is that by day 18, half of the mobile instances of application X crash and half stay up and running. This implementation computes the median survival time in the method GetMedianSurvivalTime.

Another question you can answer using the KM estimates is whether there's a difference in the survivability of two (or more) different applications. One way to approach this question is to visually plot the KM estimates corresponding to each application. This type of plot is described in **Figure 7**, and compares the estimated survival functions of applications X and Y.



```
Computing KM survival probability estimates

Kaplan Meier estimates computed

Kaplan Meier estimate for Application X
Time     Survival Probability
0 days   1.000
5 days   0.889
12 days  0.711
18 days  0.474
22 days  0.000


Kaplan Meier estimate for Application Y
Time     Survival Probability
0 days   1.000
1 days   0.900
5 days   0.800
6 days   0.700
10 days  0.583
11 days  0.467
20 days  0.311
22 days  0.156
```

Figure 5 **Survival Analysis Demo Output for KM Estimates for Applications X and Y**

The green curve represents survival function of application X and the blue curve represents survival function of application Y.

From the plot, you can see the survival function of application X tops the survival function of application Y. Therefore, you can infer application X has better survivability than application Y and, thus, is more robust.

While visualizing survival functions may help in determining survivability differences, some cases are not as clear-cut. Fortunately, there's a statistical approach to test for such differences in a formal and rigorous way, called the Log Rank Test. This is an algorithm that tests whether there's a significant difference between two (or more) survival distributions in a non-parametric way. The SA literature includes a detailed discussion about this and most SA statistical libraries include Log Rank test implementations.

It's worth noting there's another popular algorithm to estimate the survival function in a non-parametric way called the Nelson-Aalen (NA) estimator. The NA estimates the cumulative hazard function from survival data. You can then derive the survival function from this estimate using a mathematical formula that ties it to the cumulate hazard function. You can find more details about this estimator in the SA literature.

Wrapping Up

I've introduced basic concepts and terminology from the statistical branch of survival analysis. I showed how to implement the non-parametric Kaplan-Meier estimator and applied it to an example comparing the robustness of mobile applications. This estimator can help determine whether there's a difference in the survivability of the two applications. I also mentioned a rigorous statistical test to check for differences, called the Log Rank test. Another quantity I derived using the KM estimator is the median survival time, which also points to survivability differences between applications X and Y.

Finally, I mentioned the Nelson-Aalen estimator as an alternative non-parametric method for estimating the survival function. It doesn't directly estimate the survival function like the KM estimator, but rather estimates the cumulative hazard function. You can then derive the survival function from the cumulate hazard function.

> While visualizing survival functions may help in determining survivability differences, some cases are not as clear-cut.

This only scratches the surface of the rich field of SA. The applications span areas from medicine to engineering and whose methods and algorithms are implemented in many statistical packages. With the proliferation of mobile applications and Software as a Service enterprise deployments, I anticipate SA methods can play a role in monitoring and improving the quality of such deployments. ■

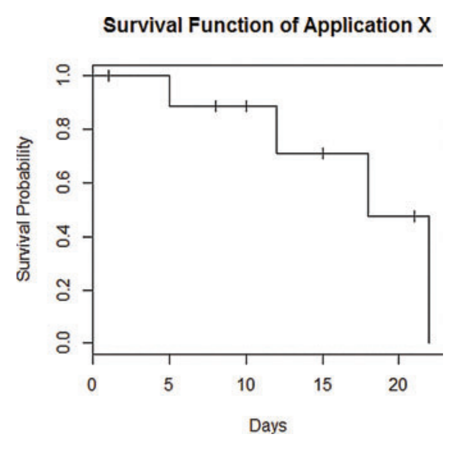


Figure 6 **KM Estimate of the Survival Function for Mobile Application X**

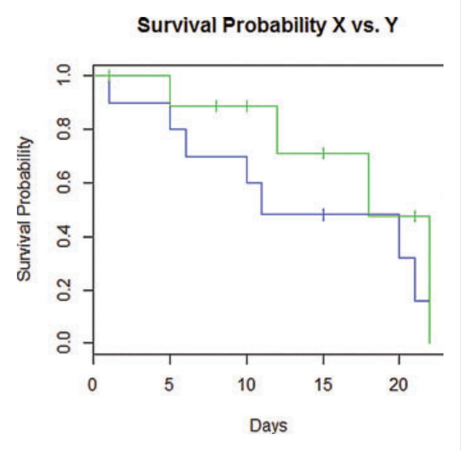


Figure 7 **KM Estimates for Mobile Applications X and Y**

Zvi Topol *works as a senior scientist in marketing analytics in New York City. He designs and applies non-linear large-scale optimization algorithms and statistical methods to improve marketing planning for big Fortune 500 companies.*

Thanks to the following technical expert for reviewing this article: *Dr. James McCaffrey (Microsoft Research)*

What's New in Windows 8.1 for Windows Store Developers

Windows 8.1 is a substantial update to the Windows OS, with many new enhancements and features to help you innovate and build the best creations possible. In this article, I'll look at the new enhancements in Windows 8.1 for developers who build Windows Store apps.

More Ways to Do Windows and Tiles

Before Windows 8.1, your app could display in one of three modes: full view (landscape or portrait), filled view or snapped view. Users of Surfaces and other devices requested more control over window management such as, for example, being able to view more than two apps concurrently. Therefore, to reflect the varied uses of the customer base, the Windows team responded by adding more ways to manage and organize windows and screen real estate. This means users can position windows side by side equally or in any proportions they want, and an app can also create multiple views that users can size and position independently.

Previously, in Windows 8 apps built with JavaScript, you'd use CSS media queries to control how the page lays itself out depending on its view state (full, filled or snapped). In Windows 8.1, this has changed so you only need CSS media queries that target the dimensions and orientations of the screen, as shown here:

```
@media screen and (min-width: 500px) and (max-width: 1023px) {
  /* CSS styles to change layout based on window width */
}
@media (min-width: 1024px) {
  /* CSS styles to change layout based on window width */
}
@media screen and (orientation: portrait) {
  /* CSS styles to change layout based on orientation */
}
```

This means you don't need to adjust or query for specific app view states, as you did before. You only need to use media queries and set a minimum width and orientation, which you can do in the media query itself. The mandatory minimum height of any Windows app is 768 pixels.

Tiles usually bring the user to the app in the first place. Located in the Start menu, live tiles are an excellent modern feature of Windows and Windows Phone. No other platform has quite the same capability to show all your data in a real-time dashboard the way Windows does. That said, as a developer, you can extend your apps to use four different tile sizes: small, medium, wide and large, as shown in **Figure 1**.

The package manifest in Visual Studio contains a Visual Assets tab where you can configure the tile sizes, along with other visual assets such as the splash screen.

New Visual Studio 2013 Project Templates Help You Build Modern Apps

As you might expect, with each Visual Studio release come new project templates. New Windows Store apps built with JavaScript (using the Windows Library for JavaScript, or WinJS) project templates include a Hub template, while the new XAML project templates include Hub, Portable Class Library and Coded UI Test.

The new Hub project template in both WinJS and XAML encapsulates a popular design approach I refer to as "modern." Its default layout contains five different sections carefully crafted so you can offer varied visual arrangements of data to your users. The Hub layout makes it easier for users to scan and pinpoint what's important to them. Designing a modern UI means you present data in ways different from previous non-modern, traditional techniques, with a focus on the user and usability. The Hub project does just that.

Inside the Hub project's pages folder live three folders named hub, item and section. Each of these comes with its own corresponding .html, .js and .css files. In the XAML projects, there are equivalent pages named HubPage.xaml, SectionPage.xaml and ItemPage.xaml in the root folder. **Figure 2** shows what the Hub project, featuring the Hub control, looks like at run time.

As you can see, the Hub project and control show a panorama view of nicely arranged content. It's a sleek and modern design.

Updated and New HTML and XAML Controls for a Modern UI

New controls and control improvements in all project types make them simpler to use. These new and updated controls make it easier than ever to create and publish a modern app. In both HTML and XAML, there are performance and data-binding control enhancements. For a primer on Windows Store controls, see my *MSDN Magazine* article, "Mastering Controls and Settings in Windows Store Apps Built with JavaScript," at msdn.microsoft.com/magazine/dn296546.

In the Hub project template comes the Hub control, new for both WinJS and XAML. The default template's Hub control structures the UI layout with five sections to scroll through horizontally, all in the app's starting page. The hero section is the crown jewel of the app, often used for presenting the featured news story, recipe, sports score, weather data or whatever it may be. It's also the first thing a user will see after the splash screen. Provided as a starting point for the developer, the next four sections simply contain data items

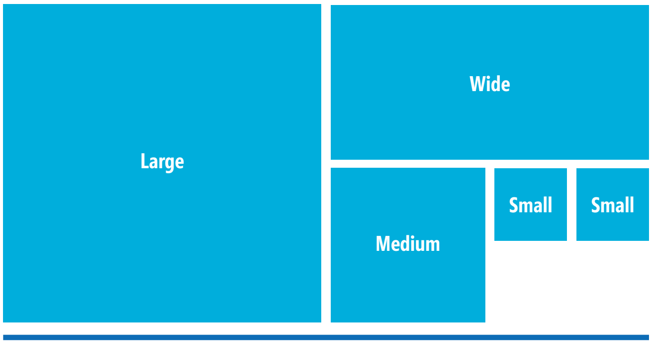


Figure 1 **New Live Tiles in Windows (Not to Scale)**

of varied sizes. Users can navigate to the listing of section 3's group membership or to individual items in section 3. Of course, the Hub control is flexible and can accommodate any number of sections with any content. It's designed to easily handle heterogeneous content of different kinds and from different sources, as opposed to strictly homogeneous content of similar data from the same source.

The Grid template relies only on the ListView control. Now the new Hub control contains an embedded ListView control so navigation works as you'd expect, to either the group listing or an individual item, depending on which item the user taps or clicks.

This ListView has many modern enhancements, including support for drag-and-drop operations. Alongside drag and drop is a ListView enhancement for reordering items. Simply set the itemsReorderable property of the ListView to true and no other code is required. The ListView includes several other enhancements, including improved cell spanning, better accessibility and better memory management.

While the ListView control has many new and shiny features, there's another control worth mentioning: the Repeater. Several UI controls across the Microsoft .NET Framework use repeating controls. For example, there's an ASP.NET Repeater control. Grid controls and the like exist throughout the .NET platform, customized to the varied ways you can build a UI with the .NET Framework. As you probably suspect, you can use the Repeater control to generate a list of items with styling from a data set. In WinJS, this means the Repeater will properly render just about any embedded HTML or WinJS controls. **Figure 3** shows an example of the Repeater control. Note how it works much like a ListView, except that groups are no longer required. As you can see in the sample data in **Figure 3**, the JavaScript creates a simple array.

The NavBar is another control that improves the UX by providing menu options in a way that's conducive to user interaction. Unlike the JavaScript menus on popular Web sites from days of yore, modern menu items are large and optimized for a variety of input devices. We've all seen someone not so skilled with a mouse struggle with those tiny, cascading Web site menus of the past. This means, as part of modern app UI design principles, the NavBar works well with touch input, a must-have feature for tablets. The user invokes the navigation toolbar by swiping from the top or bottom edges, using the Windows key+Z shortcut or by a right-click. If you've used the AppBar control, the NavBar control works almost identically.

Those who want to integrate a modern Web site with client apps can use the new WebView control. It can present data from the Internet much easier than in previous WinJS versions that used an iframe. The WebView control is an HTML element that looks like this:

```
<x-ms-webview id="webview" src="http://rachelappel.com"
  style="width: 400px; height: 400px;"></x-ms-webview>
```

This is different from the standard way to create WinJS controls by using a <div> element and setting the data-win-options attribute. The WebView control serves as a container that's hosting the external content. Along with that are security and sandboxing implications, so using an HTML element works better than a typical control in this case. WebView isn't a control that you can just implement with other HTML elements, and it must be supported directly in the app host. Note that work is being done to propose a <webview> element as a standard for consideration by the World Wide Web Consortium (W3C).

Until this Windows release, XAML hasn't had parity with HTML as far as controls were concerned. Now, though, XAML has caught up by adding the following new controls:

- **AppBar:** The menu bar across the bottom of the screen.
- **CommandBar:** An individual menu bar item.

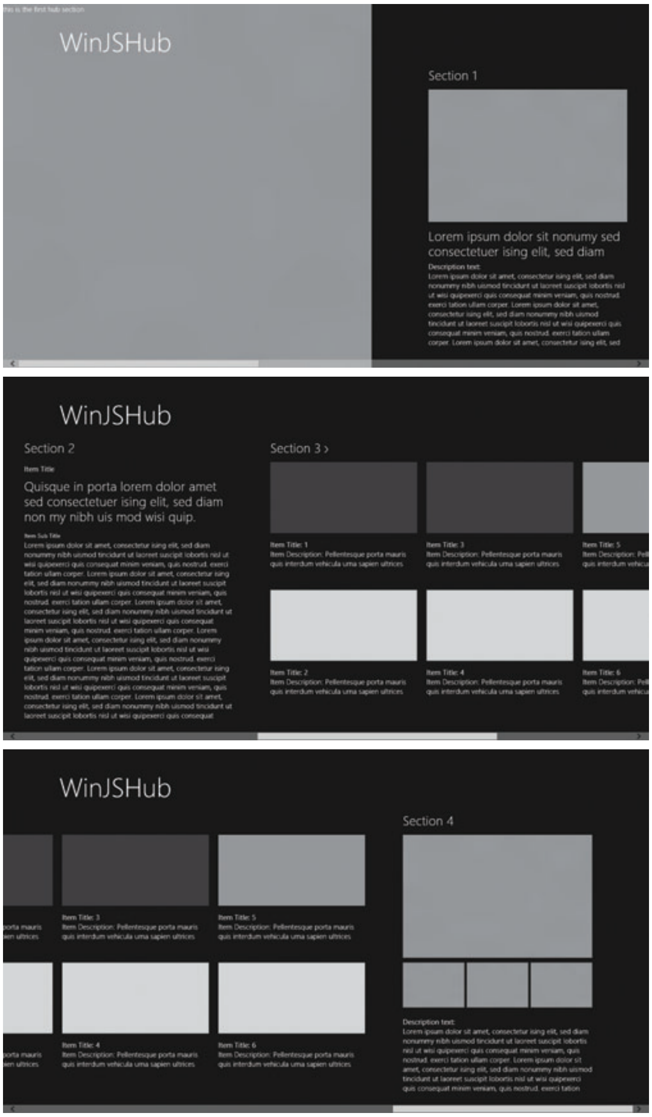

This is the first hub section
WinJSHub
Section 1
Lorem ipsum dolor sit nonumy sed consectetuer ising elit, sed diam
Description text:
WinJSHub
Section 2
Item Title
Quisque in porta lorem dolor amet sed consectetuer ising elit, sed diam non my nibh uis mod wisi quip.
Item Sub Title
Section 3 ›
Item Title: 1
Item Description: Pellentesque porta mauris quis interdum vehicula urna sapien ultrices
Item Title: 3
Item Description: Pellentesque porta mauris quis interdum vehicula urna sapien ultrices
Item Title: 5
Item Title: 2
Item Description: Pellentesque porta mauris quis interdum vehicula urna sapien ultrices
Item Title: 4
Item Description: Pellentesque porta mauris quis interdum vehicula urna sapien ultrices
Item Title: 6
WinJSHub
Section 4
Item Title: 3
Item Description: Pellentesque porta mauris quis interdum vehicula urna sapien ultrices
Item Title: 5
Item Description: Pellentesque porta mauris quis interdum vehicula urna sapien ultrices
Item Title: 4
Item Description: Pellentesque porta mauris quis interdum vehicula urna sapien ultrices
Item Title: 6
Item Description: Pellentesque porta mauris quis interdum vehicula urna sapien ultrices
Description text:

Figure 2 **The Hub Project at Run Time**

Figure 3 **HTML and JavaScrip That Builds a Simple Repeater Control**

```
<!—HTML -- >
<div id="listTemplate" data-win-control="WinJS.Binding.Template">
  <li data-win-bind="textContent: title"></li>
</div>
<ul data-win-control="WinJS.UI.Repeater"
  data-win-options="{data: RepeaterExample.basicList,
  template: select('#listTemplate')}">
</ul>
// JavaScript
(function () {
  "use strict";
  var basicList2 = new WinJS.Binding.List(
    [
      { title: "Item 1" },
      { title: "Item 2" },
      { title: "Item 3" },
      { title: "Item 4" }
    ]);
  WinJS.Namespace.define("RepeaterExample",
    {
      basicList: basicList2
    });
})();
```

- **DatePicker:** A set of three dropdowns that capture the date from the user.
- **Flyout:** A modeless, light-dismiss dialog box or a settings control.
- **Hub:** This lets you display a panorama of various-sized items in groups.
- **Hyperlink:** A hyperlink that navigates to a URI.
- **MenuFlyout:** A predefined Flyout specifically styled so it displays a list of menu items.
- **SettingsFlyout:** A Flyout that appears from the right side of the screen upon the user's swipe or interaction. It's used for managing app settings.
- **TimePicker:** A control that lets the user select hours, minutes, seconds and other segments of time. It often complements the DatePicker.

Now there's less need for XAML developers to create their own visual elements, as many are now part of the UI framework.

Windows 8.1 Has More Choice in Search

Windows 8 introduced the concept of a charm—a shortcut to a common task. Users have their habits, and search as a way to launch apps or find information is a common one. Users frequently search for information, as search engine results will attest. Search is such an important part of computing that it's part of the Windows OS and now there's a search control to complement the charms. When you have data local to your app that users should be able to search, use the SearchBox control, and when they need to do a wider-scoped or Internet search, go with the SearchPane (the Search Windows charm introduced in Windows 8). After you configure the search contract in package.appmanifest in the declarations tab, you can provide search services to your users. You may have either the SearchBox or the SearchPane in your app, but not both.

Adding a SearchBox control is as easy as applying the data-win-control attribute to WinJS.UI.SearchBox, as you might expect:

```
<div id="searchBoxId"
  data-win-control="WinJS.UI.SearchBox"
  data-win-options="{focusOnKeyboardInput: true}">
</div>
```

XAML keeps the SearchBox class definition in the Windows.UI.Xaml.Controls namespace, and the declarative syntax looks like this:

```
<SearchBox x:Name="SearchBox"
  FocusOnKeyboardInput="True"
  QuerySubmitted="SearchBox_QuerySubmitted"
  Height="35" />
```

Microsoft recommends adding instant search to your apps. Instant search is when users simply type to activate and start a search query. Go to the Windows 8 Start screen and just start typing. You'll notice the SearchPane immediately initiates a search query across the device as well as on the Internet. You can, of course, emulate this behavior in your apps like the preceding code samples do by setting the HTML data-win-option attribute to focusOnKeyboardInput or the XAML FocusOnKeyboardInput value to True.

Use Contact and Calendar APIs to Stay Connected and Up-to-Date

With Windows 8.1, you get convenient APIs to interact with a user's contacts and calendar if the user allows. The Contacts API enables a source app to query the data store by e-mail address or phone number and provide relevant information to the user. The Calendar API allows you to add, replace, remove, or otherwise work with appointments and show the user's default appointment provider app (for example, the built-in calendar app or Outlook) on screen next to your app at run time. This means your app can seamlessly integrate with the built-in apps.

In the Contact API, Windows 8.1 uses the Windows.ApplicationModel.Contacts.Contact class to represent a contact, and the older ContactInformation class used in Windows 8 has been deprecated. Luckily, the API documentation clearly labels each member of deprecated namespaces with the following message: "<member name> may be altered or unavailable for releases after Windows 8.1," so it's easy to avoid using these. **Figure 4** shows just how easy it is to capture an e-mail address and phone number and shows a user's contact card from that data. With a bit more code, you could save or show the contact in another part of the app.

As you can see, the Contacts API is simple to use, yet allows a deep level of integration with Windows. The Calendar API is quite similar. In code, you make instances of appointment objects and

Figure 4 **Showing a Contact Card**

```
<label for="emailAddressInput">Email Address</label>
<input id="emailAddressInput" type="text"/>
<label for="phoneNumberInput">Phone Number</label>
<input id="phoneNumberInput" type="text" />
<button id="addToContactsButton" onclick=
  "addContactWithDetails()">Add to Contacts</button>
function showContactWithDetails() {
  var ContactsAPI = Windows.ApplicationModel.Contacts;
  var contact = new ContactsAPI.Contact();

  var email = new ContactsAPI.ContactEmail();
  email.address = document.querySelector("#emailAddressInput");
  contact.emails.append(email);

  var phone = new ContactsAPI.ContactPhone();
  phone.number = document.querySelector("#phoneNumberInput");
  contact.phones.append(phone);
  ContactsAPI.ContactManager.showContactCard(
    contact, {x:120, y:120, width:250, height:250},
    Windows.UI.Popups.Placement.default);
}
```

assign values to properties that represent meeting details—such as the date and time of the meeting—and then save them. Then you have contacts and calendaring capabilities in your app.

New Networking and Security APIs

No system update would be complete without networking and security improvements. These networking enhancements will let you do more via code than ever before, yet remain secure. New in the Windows Runtime (WinRT) in the Windows.Web.Http namespace are objects and methods that connect to HTTP and REST services with more power and flexibility than is available with previous APIs such as WinJS.xhr and System.Net.HttpClient. The following code shows how to connect to a RESTful service:

```
var uri = new Uri("http://example.com/data.svc");
var httpClient = new HttpClient();
httpClient.GetStringAsync(uri).done(function () {
  // Process JSON
}, error);
function error(reason) {
  WinJS.log && WinJS.log("Oops!");
}
```

Just as with any other library, the Windows.Web.Http namespace has members that perform their duties asynchronously, and with JavaScript, that means using the "done" function that runs upon return of a promise. However, if you want up-to-date, real-time apps, you can use Windows.Web.Http for standby apps that run in the background. Also note that you have all kinds of other capabilities with Windows.Web.Http, such as the ability to control the cache, control cookies, make other kinds of requests and insert filters in the pipeline to do all kinds of interesting things that I don't have room to explore here.

The good news is if you access REST services that require user credentials, you (as a user) can now manage them as multiple accounts in the Settings charm. Along with these security and account management features comes the option to use fingerprint authentication in your modern apps using the Windows Fingerprint (biometric) authentication APIs.

Modern Apps Are All About Diverse Devices

You don't get more modern than 3D printing. Windows 8.1 has it, and you can develop with it! That's not to mention the catalog of hardware- and sensor-capable APIs that are now available, including the following:

- **Human Interface Devices (HID):** A protocol that fosters communication and programmability between hardware and software.
- **Point of Service (PoS):** A vendor-neutral API for Windows Store apps that can access devices such as barcode scanners or magnetic-stripe readers.
- **USB:** Enables communication with standard USB devices.
- **Bluetooth:** Enables communication with standard Bluetooth devices.
- **3D printing:** These are C++ extensions of the 2D printing support that serve as the basis for 3D printer support. You can access Windows printing to send formatted-for-3D content to the printer through an app.
- **Scanning:** Enables support for scanners.

The preceding APIs all enable hardware peripheral integration. Since Windows 8, though, apps in both HTML and XAML have been able to take advantage of hardware integration for working with the webcam, accelerometer, pen, touch and other peripherals.

Windows 8.1 includes a set of Speech Synthesis, or text-to-speech, APIs. Using these APIs, you can transform textual data into a vocal stream—and this entails less code than you might expect. For example, the following code sample shows that once a new instance of the SpeechSynthesizer exists, you can call its synthesizeTextToStreamAsync method. The synthesizeTextToStreamAsync method accepts textual data that it then transforms into a voice stream, and then it sends that stream to a player:

```
var audio = new Audio();
var synth = new Windows.Media.SpeechSynthesis.SpeechSynthesizer();
var input = document.querySelector("#input");
synth.synthesizeTextToStreamAsync(input).then(function (markersStream) {
  var blob = MSApp.createBlobFromRandomAccessStream(
    markersStream.ContentType, markersStream);
  audio.src = URL.createObjectURL(blob, { oneTimeOnly: true });
  audio.play();
});
```

In addition to working with simple textual data, you can use the W3C standard Speech Synthesis Markup Language (SSML) for sentence construction and lexical clarification. Using this XML-based language lets you perform input and output synthesis in a more clearly defined manner, which makes a difference to the user.

Wrapping Up with New App Store Packaging Features

You can configure resources such as tile images and localized strings in the package manifest, which has changed slightly to reflect new image sizes and other configuration options. One such option is to create bundles. Bundles primarily let you add and manage locale-specific information so you can deploy your app to various geographic areas.

When you deploy your app to the store, notice there are a few changes, including an enhanced UI at the developer portal. Users can find your app easier than ever before now that Bing integrates neatly into the OS. With Bing integration, users can discover your app (or any file) via the Windows Store or via a Web site or search. In addition, apps that users install now will automatically update unless users turn the automatic update feature off. You don't need to worry about frequent app updates on the user's behalf.

I don't have enough room to list all the new and enhanced features in the Windows Runtime and Visual Studio here. I do highly suggest you review other new features such as DirectX, which sports several updates that you can read about at bit.ly/1nOpODs. In addition, Charles Petzold authors an *MSDN Magazine* column focused on DirectX, so you can expect to see more details about the new features there (bit.ly/1c37bLI). Finally, all the information you need about what to expect in Windows 8.1 is in the Windows 8.1 Feature Guide at bit.ly/1cBHgxu. ■

RACHEL APPEL *is a consultant, author, mentor and former Microsoft employee with more than 20 years of experience in the IT industry. She speaks at top industry conferences such as Visual Studio Live!, DevConnections, MIX and more. Her expertise lies within developing solutions that align business and technology focusing on the Microsoft dev stack and open Web. For more about Appel, visit her Web site at rachelappel.com.*

THANKS to the following technical expert for reviewing this article: *Kraig Brockschmidt (Microsoft)*

YOUR GUIDE TO THE .NET DEVELOPMENT UNIVERSE

This May, developers, software architects, engineers, and designers will blast off in the windy city for four days of unbiased and cutting-edge education on the Microsoft Platform. Live long and code with .NET gurus, launch ideas with industry experts and rub elbows with Microsoft stars in pre-conference workshops, 60+ sessions and fun networking events – all designed to make you better at your job. Plus, explore hot topics like Web API, jQuery, MongoDB, SQL Server Data Tools and more!

Tracks Include:

- Visual Studio/.NET Framework
- Windows Client
- JavaScript/HTML5 Client
- ASP.NET
- Cloud Computing
- Windows Phone
- Cross-Platform Mobile Development
- SharePoint
- SQL Server

CHICAGO 2014

May 5 – 8 | Chicago Hilton

Sessions Are Filling Up Quickly – Register Today!

Use promo code VSLAPR4

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

TURN THE PAGE FOR MORE EVENT DETAILS →

vslive.com/chicago

Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS

> "SEVERAL OF THE PRESENTATIONS WERE CUTTING EDGE – THEY WOULD HAVE INSIDER TIPS THAT YOU CAN'T EASILY SEARCH FOR OR WOULDN'T KNOW TO LOOK FOR."
>
> *John Kilic*
> *Web Application Developer*
> *Grand Canyon University*

AGENDA AT-A-GLANCE

Visual Studio / .NET Framework	Windows Client	Cloud Computing	Windows Phone	Cross-Platform Mobile Development

START TIME	END TIME	Visual Studio Live! Pre-Conference Workshops: Monday,	
7:30 AM	9:00 AM	Pre-Conference Workshop Registration	
9:00 AM	6:00 PM	**MW01** - Workshop: Modern UX Design - *Billy Hollis*	
6:00 PM	9:00 PM	Dine-A-Round Dinner	

START TIME	END TIME	Visual Studio Live! Day 1: Tuesday, May 6, 2014	
7:00 AM	8:00 AM	Registration	
8:00 AM	9:00 AM	**Keynote:** Modern Application Lifecycle Management - *Craig Kitterman, Group*	
9:15 AM	10:30 AM	**T01** - What's New in WinRT Development - *Rockford Lhotka*	**T02** - What's New in the Visual Studio 2013 IDE
10:45 AM	12:00 PM	**T05** - What's New for XAML Windows Store Apps - *Ben Dewey*	**T06** - ALM with Visual Studio 2013 and Team Foundation Server 2013 - *Brian Randell*
12:00 PM	1:30 PM	Lunch - Visit Exhibitors	
1:30 PM	2:45 PM	**T09** - Interaction Design Principles and Patterns - *Billy Hollis*	**T10** - What's New for Web Developers in Visual Studio this Year? - *Mads Kristensen*
3:00 PM	4:15 PM	**T13** - Applying UX Design in XAML - *Billy Hollis*	**T14** - Why Browser Link Changes Things, and How You Can Write Extensions? - *Mads Kristensen*
4:15 PM	4:45 PM	Networking Break - Visit Exhibitors	
4:45 PM	6:00 PM	**T17** - What's New for HTML/WinJS Windows Store Apps - *Ben Dewey*	**T18** - Katana, OWIN, and Other Awesome Codenames: What's coming? - *Howard Dierking*
6:00 PM	7:30 PM	Exhibitor Welcome Reception	

START TIME	END TIME	Visual Studio Live! Day 2: Wednesday, May 7, 2014	
7:00 AM	8:00 AM	Registration	
8:00 AM	9:00 AM	**Keynote:** There's No Future in the Past: A Critique of Conventional Thinking on the	
9:15 AM	10:30 AM	**W01** - Windows 8 HTML/JS Apps for the ASP.NET Developer - *Adam Tuliper*	**W02** - Creating Data-Driven Mobile Web Apps with ASPNET MVC and jQuery Mobile - *Rachel Appel*
10:45 AM	12:00 PM	**W05** - Developing Awesome 3D Applications with Unity and C#/JavaScript - *Adam Tuliper*	**W06** - Getting Started with Xamarin - *Walt Ritscher*
12:00 PM	1:30 PM	Birds-of-a-Feather Lunch & Exhibitor Raffle at 1:15pm MUST be present to win	
1:30 PM	2:45 PM	**W09** - What's New in WPF 4.5 - *Walt Ritscher*	**W10** - Building Multi-Platform Mobile Apps with Push Notifications - *Nick Landry*
3:00 PM	4:15 PM	**W13** - Implementing M-V-VM (Model-View-View Model) for WPF - *Philip Japikse*	W14 - Getting Started with Windows Phone Development - *Nick Landry*
4:30 PM	5:45 PM	**W17** - Build Maintainable Windows Store Apps with MVVM and Prism - *Brian Noyes*	W18 - Build Your First Mobile App in 1 Hour with Microsoft App Studio - *Nick Landry*
7:00 PM	9:00 PM	Blues after Dark at Buddy Guy's Legends	

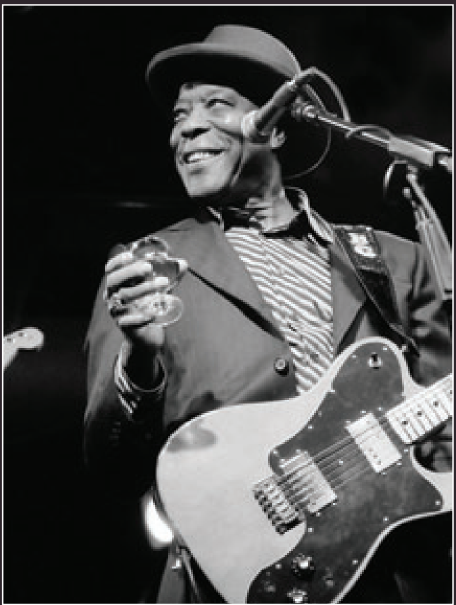
START TIME	END TIME	Visual Studio Live! Day 3: Thursday, May 8, 2014	
7:30 AM	8:00 AM	Registration	
8:00 AM	9:15 AM	**TH01** - Leveraging Windows Azure Web Sites (WAWS) - *Rockford Lhotka*	**TH02** - To Be Announced
9:30 AM	10:45 AM	**TH05** - Zero to Connected with Windows Azure Mobile Services - *Brian Noyes*	**TH06** - Essential C# 6.0 - *Mark Michaelis*
11:00 AM	12:15 PM	**TH09** - Building Services with ASP.NET MVC Web API Deep Dive - *Marcel de Vries*	**TH10** - Performance and Diagnostics Hub in Visual Studio 2013 - *Brian Peek*
12:15 PM	1:30 PM	Lunch	
1:30 PM	2:45 PM	**TH13** - Beyond Hello World: A Practical Introduction to Node.js on Windows Azure Websites - *Rick Garibay*	**TH14** - Visual Studio 2013 Release Manager: Reduce Your Cycle Time to Improve Your Value Delivery - *Marcel de Vries*
3:00 PM	4:15 PM	**TH17** - From the Internet of Things to Intelligent Systems: A Developer's Primer - *Rick Garibay*	**TH18** - Create Automated Cross Browser Testing of Your Web Applications with Visual Studio CodedUI - *Marcel de Vries*
4:30 PM	5:30 PM	Conference Wrap-Up Panel: *Andrew Brust, Miguel Castro, Rockford Lhotka, Ted Neward,*	

**Speakers and sessions subject to change*

ASP.NET	JavaScript / HTML5 Client	SharePoint	SQL Server
May 5, 2014 (Separate entry fee required)			
MW02 - Workshop: Data-Centric Single Page Applications with Angular, Breeze, and Web API - *Brian Noyes*		**MW03** - Workshop: SQL Server for Developers - *Andrew Brust & Leonard Lobel*	
Product Manager, Visual Studio Team, Microsoft			
T03 - HTML5 for Better Web Sites - *Robert Boedigheimer*		**T04** - Introduction to Windows Azure - *Vishwas Lele*	
T07 - Great User Experiences with CSS 3 - *Robert Boedigheimer*		T08 - Windows Azure Cloud Services - *Vishwas Lele*	
T11 - Build Angular Applications Using TypeScipt - Part 1 - *Sergey Barskiy*		**T12** - Windows Azure SQL Database – SQL Server in the Cloud - *Leonard Lobel*	
T15 - Build Angular Applications Using TypeScipt - Part 2 - *Sergey Barskiy*		**T16** - Solving Security and Compliance Challenges with Hybrid Clouds - *Eric D. Boyd*	
T19 - Building Real Time Applications with ASP.NET SignalR - *Rachel Appel*		**T20** - Learning Entity Framework 6 - *Leonard Lobel*	
Radical Changes in Our Industry - *Billy Hollis, Next Version Systems*			
W03 - Leveraging Visual Studio Online - *Brian Randell*		**W04** - Programming the T-SQL Enhancements in SQL Server 2012 - *Leonard Lobel*	
W07 - JavaScript for the C# Developer - *Philip Japikse*		**W08** - SQL Server 2014: Features Drill-down - *Scott Klein*	
W11 - Build Data-Centric HTML5 Single Page Applications with Breeze - *Brian Noyes*		**W12** - SQL Server 2014 In-memory OLTP - Deep Dive - *Scott Klein*	
W15 - Knocking it Out of the Park, with Knockout.JS - *Miguel Castro*		**W16** - To Be Announced	
W19 - JavaScript: Turtles, All the Way Down - *Ted Neward*		**W20** - Building Apps for SharePoint - *Mark Michaelis*	
TH03 - What's New in MVC 5 - *Miguel Castro*		**TH04** - Excel, Power BI and You: An Analytics Superhub - *Andrew Brust*	
TH07 - What's New in Web API 2 - *Miguel Castro*		**TH08** - Big Data 101 with HDInsight - *Andrew Brust*	
TH11 - Upgrading Your Existing ASP.NET Apps - *Pranav Rastogi*		**TH12** - NoSQL for the SQL Guy - *Ted Neward*	
TH15 - Finding and Consuming Public Data APIs - *G. Andrew Duthie*		**TH16** - Git for the Microsoft Developer - *Eric D. Boyd*	
TH19 - Provide Value to Customers and Enhance Site Stickiness By Creating an API - *G. Andrew Duthie*		**TH20** - Writing Asynchronous Code Using .NET 4.5 and C# 5.0 - *Brian Peek*	
& Brian Peek			

CHICAGO 2014

May 5 – 8 | Chicago Hilton



Visual Studio Live! Chicago Blues After Dark Reception at Buddy Guy's Legends

CONNECT WITH VISUAL STUDIO LIVE!

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

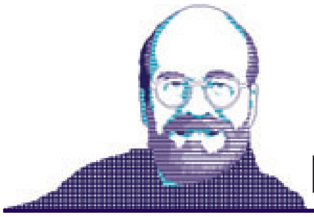
linkedin.com – Join the "Visual Studio Live" group!

SESSIONS ARE FILLING UP QUICKLY – REGISTER TODAY!

Use Promo Code VSLAPR4

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

vslive.com/chicago



3D Transforms on 2D Bitmaps

Three-dimensional graphics programming is mostly a matter of creating optical illusions. Images are rendered on a flat screen consisting of a two-dimensional array of pixels, but these objects must appear to have a third dimension with depth.

Probably the biggest contributor to the illusion of 3D is shading—the art and science of coloring pixels so that surfaces resemble real-world textures with lighting and shadows.

Underneath all that, however, is an infrastructure of virtual objects described by 3D coordinates. Ultimately, these 3D coordinates are flattened into a 2D space, but until that final step, 3D coordinates are often systematically modified in various ways through the use of transforms. Mathematically, transforms are operations in matrix algebra. Achieving a fluency in these 3D transforms is crucial for anyone who wants to become a 3D graphics programmer.

Recently, I've been exploring the several ways 3D is supported in the Direct2D component of DirectX. Exploring 3D within the relative familiarity and comfort of Direct2D allows you to become acquainted with 3D concepts prior to the very scary deep plunge into Direct3D.

Bitmaps in 3D?

One of the several ways 3D is supported in Direct2D is tucked away as the last argument to the DrawBitmap methods defined by ID2D1DeviceContext. This argument lets you apply a 3D transform to a 2D bitmap. (This feature is special to ID2D1DeviceContext. It is *not* supported by the DrawBitmap methods defined by ID2D1RenderTarget or the other interfaces that derive from that.)

The DrawBitmap methods defined by ID2D1DeviceContext have a final argument of type D2D1_MATRIX_4X4_F, which is a 4×4 transform matrix that performs a 3D transform on the bitmap as it's rendered to the screen:

```
void DrawBitmap(ID2D1Bitmap *bitmap,
                D2D1_RECT_F *destinationRectangle,
                FLOAT opacity,
                D2D1_INTERPOLATION_MODE interpolationMode,
                const D2D1_RECT_F *sourceRectangle,
                const D2D1_MATRIX_4X4_F *perspectiveTransform)
```

This appears to be sole purpose of D2D1_MATRIX_4X4_F. It isn't used elsewhere in DirectX. In Direct3D programming, the DirectX Math library is used instead to represent 3D transforms.

DD2D1_MATRIX_4X4_F is a typedef for D2D_MATRIX_4X4_F, which is defined in **Figure 1**. It's basically a collection of 16 float values arranged in four rows of four columns. You can reference the value in the third row and second column using the data member _32, or you can get at the same value as the zero-based array element m[2][1].

Code download available at msdn.microsoft.com/magazine/msdnmag0414.

Three-dimensional graphics programming is mostly a matter of creating optical illusions.

However, when showing you the mathematics of the transform, I'll instead refer to the element in the third row and second column of the matrix as m32. The entire matrix can be represented in traditional matrix notation, like so:

$$\begin{vmatrix} m11 & m12 & m13 & m14 \\ m21 & m22 & m23 & m24 \\ m31 & m32 & m33 & m34 \\ m41 & m42 & m43 & m44 \end{vmatrix}$$

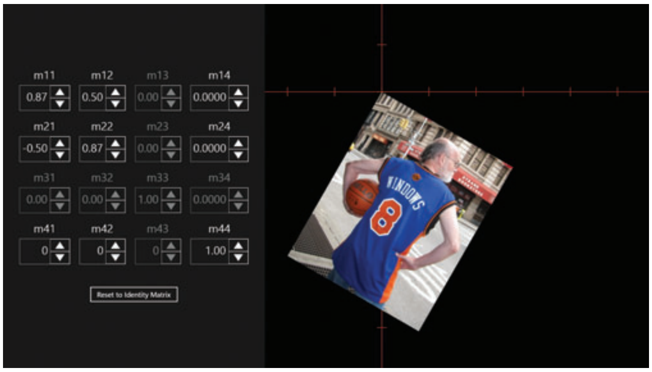
The 3D bitmap transform in Direct2D underlies a similar facility in the Windows Runtime, where it shows up as the Matrix3D structure and the Projection property defined by UIElement. The two mechanisms are so similar that you can cross-fertilize your knowledge and experience between the two environments.

The Linear Transform

Many people encountering 3D transforms for the first time ask: Why is it a 4×4 matrix? Shouldn't a 3×3 matrix be adequate for 3D?

Figure 1 **The 3D Transform Matrix Applied to Bitmaps**

```
typedef struct D2D_MATRIX_4X4_F
{
  union
  {
    struct
    {
      FLOAT _11, _12, _13, _14;
      FLOAT _21, _22, _23, _24;
      FLOAT _31, _32, _33, _34;
      FLOAT _41, _42, _43, _44;
    } ;
    FLOAT m[4][4];
  };
} D2D_MATRIX_4X4_F;
```



m11	m12	m13	m14
0.87	0.50	0.00	0.0000
m21	**m22**	**m23**	**m24**
-0.50	0.87	0.00	0.0000
m31	**m32**	**m33**	**m34**
0.00	0.00	1.00	0.0000
m41	**m42**	**m43**	**m44**
0	0	0	1.00

Figure 2 **The BitmapTransformExperiment Program**

To answer this question while exploring the 3D transform on DrawBitmap, I created a program named BitmapTransformExperiment that's included with the downloadable code for this article. This program contains homemade spinner controls that let you select values for the 16 elements of the transform matrix and see how the matrix affects the display of a bitmap. **Figure 2** shows a typical display.

For your initial experimentations, restrict your attention to the top three rows and the leftmost three columns of the matrix. These make up a 3×3 matrix that performs the following transform:

$$\begin{vmatrix} x & y & z \end{vmatrix} \times \begin{vmatrix} m11 & m12 & m13 \\ m21 & m22 & m23 \\ m31 & m32 & m33 \end{vmatrix} = \begin{vmatrix} x' & y' & z' \end{vmatrix}$$

The 1×3 matrix at the left represents a 3D coordinate. For the bitmap, the x value ranges from 0 to the bitmap width; y ranges from 0 to the bitmap height; and z is 0.

When the 1×3 matrix is multiplied by the 3×3 transform matrix, the standard matrix multiplication results in transformed coordinates:

$$x' = m11 \cdot x + m21 \cdot y + m31 \cdot z$$
$$y' = m12 \cdot x + m22 \cdot y + m32 \cdot z$$
$$z' = m13 \cdot x + m23 \cdot y + m33 \cdot z$$

The identity matrix—in which the diagonal elements of m11, m22, and m33 are all 1 and everything else is 0—results in no transform.

Because I'm starting out with a flat bitmap, the z coordinate is 0, hence m31, m32, and m33 have no effect on the result. When the transformed bitmap is rendered to the screen, the (x′, y′, z′) result is collapsed onto a flat 2D coordinate system by ignoring the z′ coordinate, which means that m13, m23, and m33 have no effect. This is why the third row and third column are grayed out in the BitmapTransformExperiment program. You can set values for these matrix elements, but they don't affect how the bitmap is rendered.

What you'll discover is that m11 is a horizontal scaling factor with a default value of 1. Make it larger or smaller to increase or decrease the bitmap width, or make it negative to flip the bitmap around the vertical axis. Similarly, m22 is a vertical scaling factor.

The m21 value is a vertical skewing factor: Values other than 0 turn the rectangular bitmap into a parallelogram as the right edge is shifted up or down. Similarly, m12 is a horizontal skewing factor.

A combination of horizontal skewing and vertical skewing can result in rotation. To rotate the bitmap clockwise by a particular angle, set m11 and m22 to the cosine of that angle, set m21 to the sine of the angle, and set m12 to the negative sine. **Figure 2** shows rotation by 30 degrees.

In the context of 3D, the rotation shown in **Figure 2** is actually rotation around the Z axis, which conceptually extends out from the screen. For an angle of α, the transform matrix looks like this:

$$\begin{vmatrix} \cos a & \sin a & 0 \\ -\sin a & \cos a & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

It's also possible to rotate the bitmap around the Y axis or the X axis. Rotation around the Y axis doesn't affect the y coordinate, so the transform matrix is this:

$$\begin{vmatrix} \cos a & 0 & -\sin a \\ 0 & 1 & 0 \\ \sin a & 0 & \cos a \end{vmatrix}$$

If you try this with the BitmapTransformExperiment program, you'll discover that only the m11 value has an effect. Setting it to the cosine of a rotation angle merely decreases the width of the rendered bitmap. That decrease in width is consistent with rotation around the Y axis.

Similarly, this is rotation around the X axis:

$$\begin{vmatrix} 1 & 0 & 0 \\ 0 & \cos a & \sin a \\ 0 & -\sin a & \cos a \end{vmatrix}$$

In the BitmapTransformExperiment program, this results in reducing the height of the bitmap.

Probably the biggest contributor to the illusion of 3D is shading—the art and science of coloring pixels so that surfaces resemble real-world textures with lighting and shadows.

The signs of the two sine factors in the transform matrices govern the direction of rotation. Conceptually, the positive Z axis is assumed to extend from the screen, and the rotations follow the left-hand rule: Align the thumb of your left hand with the axis of rotation and point it toward positive values; the curve of your other fingers indicates the direction of rotation for positive angles.

The type of 3D transform represented by this 3×3 transform matrix is known as a linear transform. The transform only involves constants multiplied by the x, y and z coordinates. No matter what numbers you enter in the first three rows and columns of the transform matrix, the bitmap is never transformed into anything more exotic than a parallelogram; in three dimensions, a cube is always transformed into a parallelepiped.

You've seen how the bitmap can be scaled, skewed and rotated, but throughout this exercise, the upper-left corner of the bitmap has remained fixed at a single location. (That location is governed by a 2D transform set in the Render method prior to the DrawBitmap call.) The inability to move the upper-left corner of the bitmap results from the mathematics of the linear transform. There's nothing in the transform formula that can shift a (0, 0, 0) point to another location, which is a type of transform known as translation.

Achieving Translation

To understand how to obtain translation in 3D, let's briefly think about 2D transforms.

In two dimensions, a linear transform is a 2×2 matrix, and it's capable of scaling, skewing and rotating. To get translation as well, the 2D graphics are assumed to exist in 3D space but on a 2D plane where the z coordinate always equals 1. To accommodate the additional dimension, the 2D linear transform matrix is expanded into a 3×3 matrix, but usually the last row is fixed:

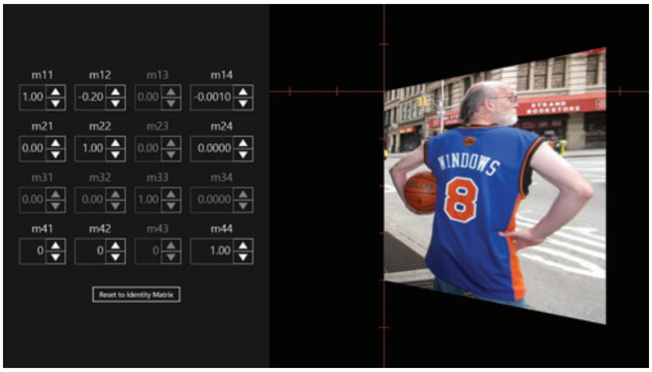
$$\begin{vmatrix} x & y & 1 \end{vmatrix} \times \begin{vmatrix} m11 & m12 & 0 \\ m21 & m22 & 0 \\ m31 & m32 & 1 \end{vmatrix}$$

The matrix multiplication results in these transform formulas:

$$x' = m11 \cdot x + m21 \cdot y + m31$$
$$y' = m12 \cdot x + m22 \cdot y + m32$$
$$z' = 1$$

The m31 and m32 factors are the translation factors. The secret behind this process is that translation in two dimensions is equivalent to skewing in three dimensions.

An analogous process is used for 3D graphics: The 3D coordinate is actually assumed to exist in 4D space where the coordinate of the fourth dimension is 1. But to represent a 4D coordinate point, you have a silly little practical problem: A 3D point is (x, y, z) and no letter comes after z, so what letter do you use for the fourth dimension? The closest available letter is w, so the 4D coordinate is (x, y, z, w).



m11	m12	m13	m14
1.00	-0.20	0.00	-0.0010
m21	**m22**	**m23**	**m24**
0.00	1.00	0.00	0.0000
m31	**m32**	**m33**	**m34**
0.00	0.00	1.00	0.0000
m41	**m42**	**m43**	**m44**
0	0	0	1.00

Figure 3 **Perspective in BitmapTransformExperiment**

The 3D linear transform matrix is expanded to 4×4 to accommodate the extra dimension:

$$\begin{vmatrix} x & y & z & 1 \end{vmatrix} \times \begin{vmatrix} m11 & m12 & m13 & 0 \\ m21 & m22 & m23 & 0 \\ m31 & m32 & m33 & 0 \\ m41 & m42 & m43 & 1 \end{vmatrix}$$

The transform formulas are now:

$$x' = m11 \cdot x + m21 \cdot y + m31 \cdot z + m41$$
$$y' = m12 \cdot x + m22 \cdot y + m32 \cdot z + m42$$
$$z' = m13 \cdot x + m23 \cdot y + m33 \cdot z + m43$$
$$w' = 1$$

You now have translation along the X, Y, and Z axes with m41, m42, and m43. The calculation seems to occur in 4D space, but it's actually restricted to a 3D cross-section of 4D space where the w coordinate is always 1.

Homogenous Coordinates

If you play around with the m41 and m42 values in BitmapTransformExperiment, you'll see they do indeed result in horizontal and vertical translation.

But what about that last row of the matrix? What happens if you *don't* restrict that last row to 0s and 1s? Here's the full 4×4 transform applied to a 3D point:

$$\begin{vmatrix} x & y & z & 1 \end{vmatrix} \times \begin{vmatrix} m11 & m12 & m13 & m14 \\ m21 & m22 & m23 & m24 \\ m31 & m32 & m33 & m34 \\ m41 & m42 & m43 & m44 \end{vmatrix}$$

The formulas for x', y' and z' remain the same, but w' is now calculated like this:

$$w' = m14 \cdot x + m24 \cdot y + m34 \cdot z + m44$$

And that's a real problem. Previously, a little trick was employed to use a 3D cross-section of 4D space where the w coordinate always equals 1. But now the W coordinate is no longer 1, and you've been propelled out of that 3D cross-section. You're lost in 4D space, and you need to get back to that 3D cross-section where w equals 1.

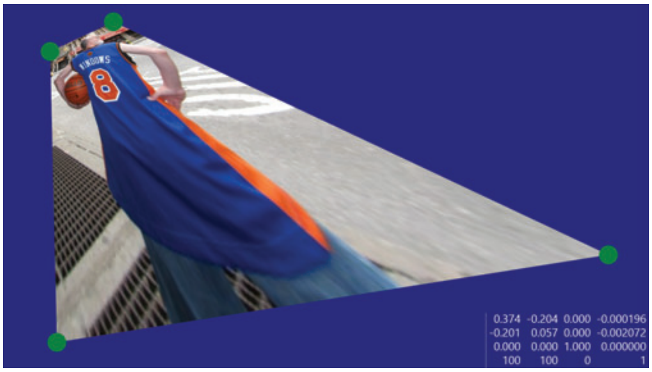
There's no need to build a trans-dimensional space-time machine, however. Fortunately, you can make the leap mathematically by dividing all the transformed coordinates by w':

$$x' = \frac{m11 \cdot x + m21 \cdot y + m31 \cdot z + m41}{m14 \cdot x + m24 \cdot y + m34 \cdot z + m44}$$

$$y' = \frac{m12 \cdot x + m22 \cdot y + m32 \cdot z + m42}{m14 \cdot x + m24 \cdot y + m34 \cdot z + m44}$$

$$z' = \frac{m13 \cdot x + m23 \cdot y + m33 \cdot z + m43}{m14 \cdot x + m24 \cdot y + m34 \cdot z + m44}$$

$$w' = \frac{m14 \cdot x + m24 \cdot y + m34 \cdot z + m44}{m14 \cdot x + m24 \cdot y + m34 \cdot z + m44} = 1$$



```
 0.374  -0.204  0.000  -0.000196
-0.201   0.057  0.000  -0.002072
 0.000   0.000  1.000   0.000000
   100     100      0          1
```

Figure 4 **The NonAffineStretch Program**

Now w′ equals 1 and you're back home!

But at what cost? You now have a division in the transform formulas, and it's easy to see how that denominator might be 0 in some circumstances. That would result in infinite coordinates.

Well, maybe that's a good thing.

When German mathematician August Ferdinand Möbius (1790–1868) invented the system I've just described (called "homogenous coordinates" or "projective coordinates"), one of his goals was to represent infinite coordinates using finite numbers.

The matrix with the 0s and 1s in the last row is called an affine transform, meaning that it doesn't result in infinity, so a transform capable of infinity is called a non-affine transform.

In 3D graphics, non-affine transforms are extremely important, for this is how perspective is achieved. Everyone knows that in real life, objects further from your eye appear to be smaller. In 3D graphics, you obtain that effect with a denominator that isn't a constant 1.

> In 3D graphics, non-affine transforms are extremely important, for this is how perspective is achieved.

Try it out in BitmapTransformExperiment: If you make m14 a small positive number—and only small values are necessary for interesting results—then values of x and y are proportionally decreased as x gets larger. Make m14 a small negative number, and larger values of x and y are increased. **Figure 3** shows that effect combined with a non-zero m12 value. The rendered bitmap is no longer a parallelogram, and the perspective suggests that the right edge has swung closer to your eyes.

Similarly, non-zero values of m24 can make the top or bottom of the bitmap seemingly swing toward you or further away. In real 3D programming, it's the m34 value that's commonly used, for that allows objects to increase or decrease in size based on their z coordinates—their distance from the viewer's eyes.

When a 3D transform is applied to 2D objects, the m44 value is usually left at 1, but it can function as an overall scaling factor. In real 3D programming, m44 is usually set to 0 when perspective is involved because conceptually the camera is at the origin. A 0 value of m44 only works if 3D objects don't have z coordinates of 0, but when working with 2D objects, the z coordinate is always 0.

Any Convex Quadrilateral

In applying this 4×4 transform matrix to a flat bitmap, you're only making use of half the matrix elements. Even so, **Figure 3** shows something you can't do with the normal two-dimensional Direct2D transform matrix, which is to apply a transform that turns a rectangle into something other than a parallelogram. Indeed, the transform objects used in most of Direct2D are called D2D1_MATRIX_3X2_F and Matrix3x2F, emphasizing the inaccessibility of the third row and the inability to perform non-affine transforms.

With D2D1_MATRIX_4X4_F, it's possible to derive a transform that maps a bitmap into any convex quadrilateral—that is, any arbitrary four-sided figure where the sides don't cross and where interior angles at the vertices are less than 180 degrees.

If you don't believe me, try playing around with the NonAffineStretch program. Note that this program is adapted from a Windows Runtime program, also called NonAffineStretch, in Chapter 10 of my book, "Programming Windows, 6th Edition" (Microsoft Press, 2013).

Figure 4 shows NonAffineStretch in use. You can use the mouse or your fingers to drag the green dots to any location on the screen. As long as you keep the figure a convex quadrilateral, a 4×4

Figure 5 **Code from RotatingTextRenderer.cpp**

```
void RotatingTextRenderer::Update(DX::StepTimer const& timer)
{
  ...
  // Begin with the identity transform
  m_matrix = Matrix4x4F();

  // Rotate around the Y axis
  double seconds = timer.GetTotalSeconds();
  float angle = 360 * float(fmod(seconds, 7) / 7);
  m_matrix = m_matrix * Matrix4x4F::RotationY(angle);

  // Apply perspective based on the bitmap width
  D2D1_SIZE_F bitmapSize = m_bitmap->GetSize();
  m_matrix = m_matrix * Matrix4x4F::PerspectiveProjection(bitmapSize.width);
}

void RotatingTextRenderer::Render()
{
  ...
  ID2D1DeviceContext* context = m_deviceResources->GetD2DDeviceContext();
  Windows::Foundation::Size logicalSize = m_deviceResources->GetLogicalSize();

  context->SaveDrawingState(m_stateBlock.Get());
  context->BeginDraw();
  context->Clear(ColorF(ColorF::DarkMagenta));

  // Move origin to top center of screen
  Matrix3x2F centerTranslation =
    Matrix3x2F::Translation(logicalSize.Width / 2, 0);

  context->SetTransform(centerTranslation *
    m_deviceResources->GetOrientationTransform2D());

  // Draw the bitmap
  context->DrawBitmap(m_bitmap.Get(),
                      nullptr,
                      1.0f,
                      D2D1_INTERPOLATION_MODE_LINEAR,
                      nullptr,
                      &m_matrix);
  ...
}
```

transform can be derived based on the dot locations. That transform is used to draw the bitmap and is also displayed in the lower-right corner. Only eight values are involved; the elements in the third row and third column are always default values, and m44 is always 1.

The mathematics behind this are a little hairy, but the derivation of the algorithm is shown in Chapter 10 of my book.

The Matrix4x4F Class

To make working with the D2D1_MATRIX_4X4_F structure a bit easier, the Matrix4x4F class in the D2D1 namespace derives from that structure. This class defines a constructor and a multiplication operator (which I used in the NonAffineStretch algorithm), and several helpful static methods for creating common transform matrices. For example, the Matrix4x4F::RotationZ method accepts an argument that is an angle in degrees and returns a matrix that represents rotation by that angle around the Z axis:

$$\begin{vmatrix} \cos a & \sin a & 0 & 0 \\ -\sin a & \cos a & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Other Matrix4x4F functions create matrices for rotation around the X and Y axes, and rotation around an arbitrary axis, which is a much more difficult matrix.

A function called Matrix4x4F::PerspectiveProjection has an argument named depth. The matrix it returns is:

$$\begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -\frac{1}{\text{depth}} \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

This means the transform formula for x' is:

$$x' = \frac{x}{1 - \frac{z}{\text{depth}}}$$

And similarly for y′ and z′, which means whenever the z coordinate equals depth, the denominator is 0, and all the coordinates become infinite.

Conceptually, this means you're viewing the computer screen from a distance of depth units, where the units are the same as the screen itself, meaning pixels or device-independent units. If a graphical object has a z coordinate equal to depth, it's depth units in front of the screen, which is right at your eyeball! That object should appear very large to you—mathematically infinite.

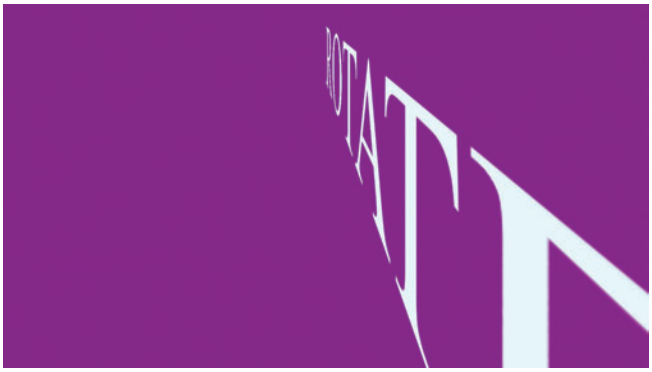


Figure 6 **The RotatingText Display**

But wait a minute: The sole purpose of the D2D1_MATRIX_4X4_F structure and the Matrix4x4F class is for calls to DrawBitmap, and bitmaps always have z coordinates of 0. So how does this m34 value of –1/depth have any effect at all?

If the PerspectiveProjection matrix is used by itself in the DrawBitmap call, it will indeed have no effect. But it's intended to be used in conjunction with other matrix transforms. Matrix transforms can be compounded by multiplying them. Although the original bitmap has no z coordinates, and z coordinates are ignored for rendering, z coordinates can certainly play a role in the compounding of transforms.

Let's look at an example. The RotatingText program creates a bitmap with the text "ROTATE" with a width that's just about half the width of the screen. Much of the Update and Render methods are shown in **Figure 5**.

In the Update method, the Matrix4x4F::RotationY method creates the following transform:

$$\begin{vmatrix} \cos a & 0 & -\sin a & 0 \\ 0 & 1 & 0 & 0 \\ \sin a & 0 & \cos a & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Multiply this by the matrix shown earlier returned from the Matrix4x4F::PerspectiveProjection method, and you'll get:

$$\begin{vmatrix} \cos a & 0 & -\sin a & \frac{\sin a}{\text{depth}} \\ 0 & 1 & 0 & 0 \\ \sin a & 0 & \cos a & \frac{-\cos a}{\text{depth}} \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

The transform formulas are:

$$x' = \frac{\cos a \cdot x}{1 + \frac{\sin a \cdot x}{\text{depth}}}$$

$$y' = \frac{y}{1 + \frac{\sin a \cdot x}{\text{depth}}}$$

These definitely involve perspective, and you can see the result in **Figure 6**.

Watch out: The depth argument to Matrix4x4F::PerspectiveProjection is set to the bitmap width, so as the rotating bitmap swings around, it might come very close to your nose. ■

CHARLES PETZOLD *is a longtime contributor to* MSDN Magazine *and the author of "Programming Windows, 6th Edition" (Microsoft Press, 2013), a book about writing applications for Windows 8. His Web site is charlespetzold.com.*

THANKS to the following Microsoft technical experts for reviewing this article: *Jim Galasyn and Mike Riches*

We Didn't Start Computing

I've always admired the April Fools' Day column as an art form. The best I've ever heard of is the April 1, 1919, cover of the British humor magazine *Punch*, which supposedly screamed: "Archduke Franz Ferdinand Found Alive! War Fought By Mistake!" (And if it didn't really say that, well, April fool, it should have.)

I've also admired Billy Joel since he delivered a superb concert at my undergrad university, back before he got famous. You've probably heard his historical song, "We Didn't Start the Fire," which uses tiny snippets to narrate history from 1950 onward: "Harry Truman, Doris Day, Red China, Johnnie Ray …." So for your amusement, I've rewritten that song to narrate the progression of our industry.

I offered to sing the song onstage at Build for only $5,000. Microsoft countered with $10,000 if I didn't. So I'm getting my revenge by publishing it here. You will probably enjoy it most if you read it while listening to the original song (bit.ly/1fnHqf0). And if you're crazy enough or masochistic enough to sing it in public, you can find an instrumental karaoke version at bit.ly/1k5RLhS. Go to it, my brave readers. As we geeks say, "a-01$_{two}$, 10$_{two}$, 11$_{two}$, 100$_{two}$ …"

Alan Turing, Frederick Brooks, John von Neumann, Donald Knuth
Ada Lovelace, Grace Hopper, Three-finger salute
Z-80, first for me, Kernighan and Ritchie C
Apple and the Macintosh in 1984

Peter Norton, Byte magazine, "The Soul of a New Machine"
PDP-8, IBM, 8-inch floppy CP/M
Bell Labs, Xerox PARC, Wang, Data General, DEC
Commodore! 64! I can't take it anymore!

We didn't start computing
It was always burning since a brain's been churning
We won't stop computing
No we won't stop it, but we'll try to top it

Bjarne Stroustrup, C++, Bill Gates, Windows, Microsoft
Minesweeper and Solitaire, Dummies tearing out their hair
World-wide-Web, Tim Berners-Lee, CompuServe and Prodigy
Dave Cutler, NT, want to strangle Clippy

Amazon and Pets.com, Y2K doesn't bomb
Circuit City, Googleplex, what the hell is ActiveX?
COM OLE and DDE, Microsoft monopoly
Janet Reno! DOJ! What else do I have to say?

We didn't start computing
It was always burning since a brain's been churning
We won't stop computing
No we won't stop it, but we'll try to top it.

Larry Page, Sergey Brin, Oracle and Ellison
Charles Petzold, Seymour Cray, Google Doodle every day
Camera phones, Nintendo Wii, Roomba Kindle Blackberry
Jeff Bezos, Craig's list, Donald Norman, no more Borland

PageMaker, Acrobat, software legend David Platt
Dell, Gateway, Lenovo, Vista is no-go
YouTube, Elon Musk, .NET Framework, Lotus Notes
Steve Jobs, iMac, iPod, iTunes, iPhone, iPad

We didn't start computing
It was always burning since a brain's been churning
We won't stop computing
No we won't stop it, but we'll try to top it

Dot-com bubble, Xbox, Google Chrome and Firefox
Steve Ballmer, David Pogue, Zynga Facebook Zuckerberg
Alan Cooper, Clippy dead, GPS and Javaheads
Jakob Nielsen, LinkedIn, Silverlight and Python
Netflix, Apple hype, PayPal Yahoo eBay Skype
XSD! USB! TLAs are BFD!

We didn't start computing
It was always burning since a brain's been churning
We won't stop computing
No we won't stop it, but we'll try to top it

Angry Birds, Snapchat, Candy Crush and Black Hat
Azure Cloud, eHarmony, Surveymonkey WebMD
Uber Quber Chatroulette, Samsung Android Babelfish
Deep Throat, Deep Blue, NSA is watching you,

Wikipedia, Instagram, Bing, IMDB and Spam
Twitter Tumblr MSN, Ballmer out Nadella in,
MOOCs, Nooks, Google Glass, Jobs and Ritchie bite the dust.
Windows 8! What's its fate? Will the next one be too late?

We didn't start computing
It's been always burning since a brains been churning
But when we are gone
Will it still burn on and on and on and on and on?

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.*

.NET TOOLS FOR DEV PROS

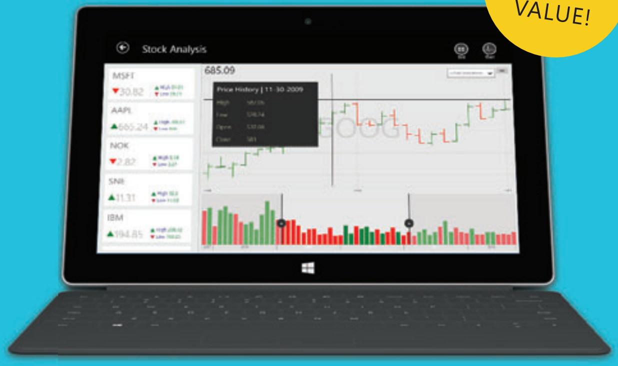
Whether you're building the most modern touch-enabled apps or maintaining and updating legacy applications, our flagship product, Studio Enterprise, helps to deliver rich, responsive, desktop and web apps on time and under budget.

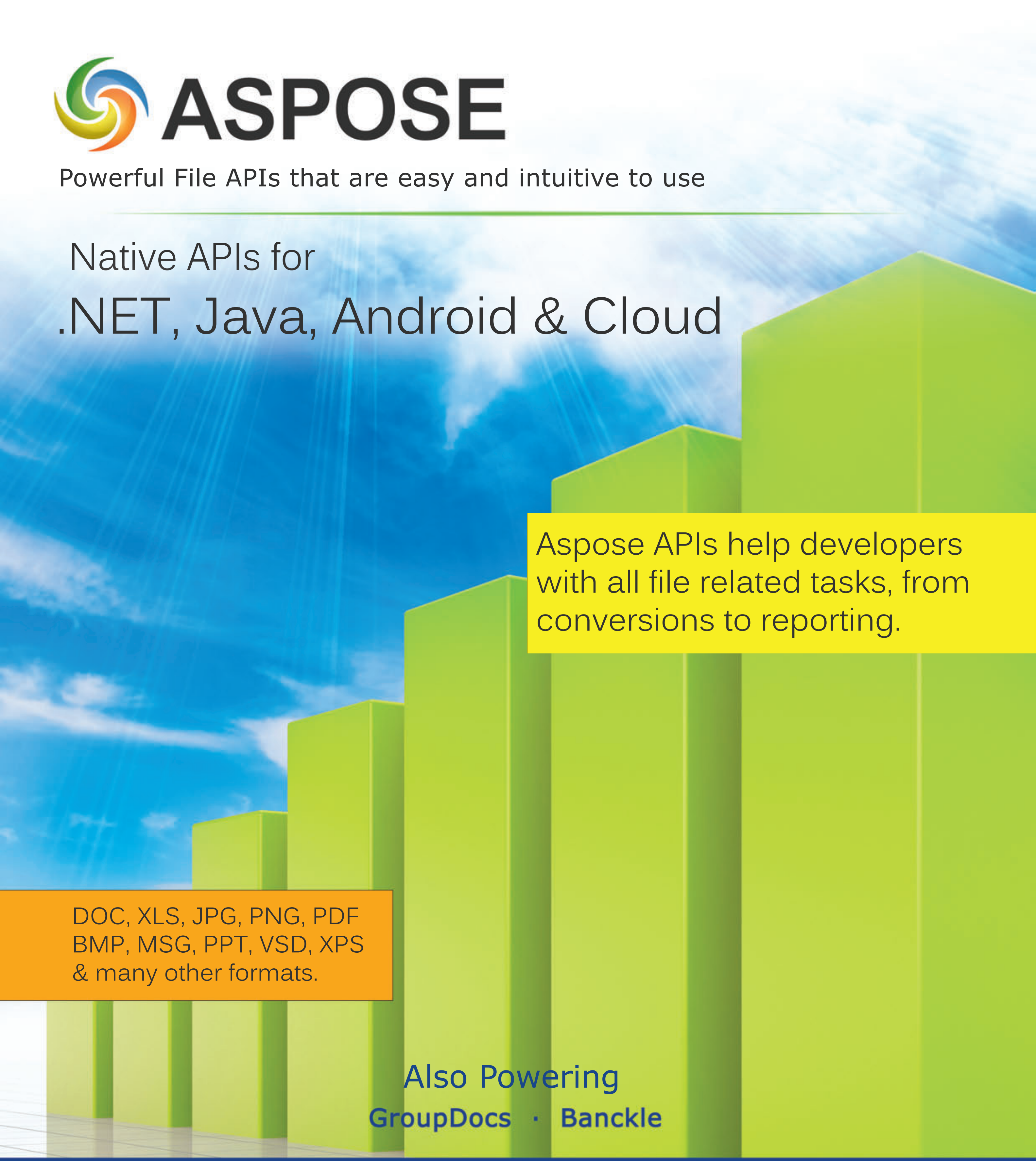
New 2014 v1 Release!

DOWNLOAD YOUR FREE TRIAL

▸ componentone.com/se

ComponentOne®
a division of GrapeCity®





DOC
PPT
XLS
PDF
MSG

Aspose.Cells

Work with spreadsheets and data without depending on Microsoft Excel

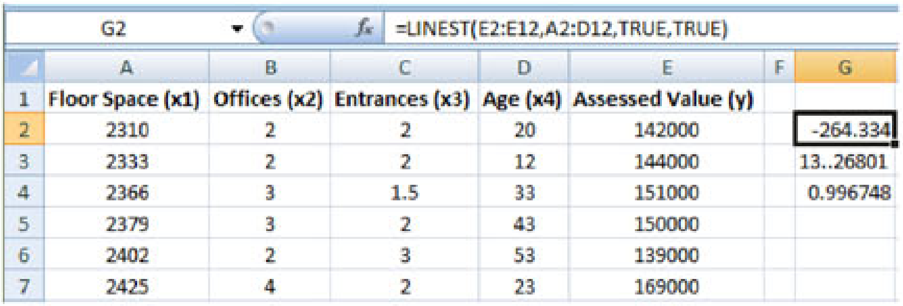
- Solution for spreadsheet creation, manipulation and conversion.
- Import and export data.

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.

A flexible API for simple and complex spreadsheet programming.

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



G2 =LINEST(E2:E12,A2:D12,TRUE,TRUE)

	A	B	C	D	E	F	G
1	Floor Space (x1)	Offices (x2)	Entrances (x3)	Age (x4)	Assessed Value (y)		
2	2310	2	2	20	142000		-264.334
3	2333	2	2	12	144000		13..26801
4	2366	3	1.5	33	151000		0.996748
5	2379	3	2	43	150000		
6	2402	2	3	53	139000		
7	2425	4	2	23	169000		

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.

www.aspose.com

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

EU: +44 141 416 1112
sales.europe@aspose.com

Oceania: +61 2 8003 5926
sales.asiapacific@aspose.com

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.

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.

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

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

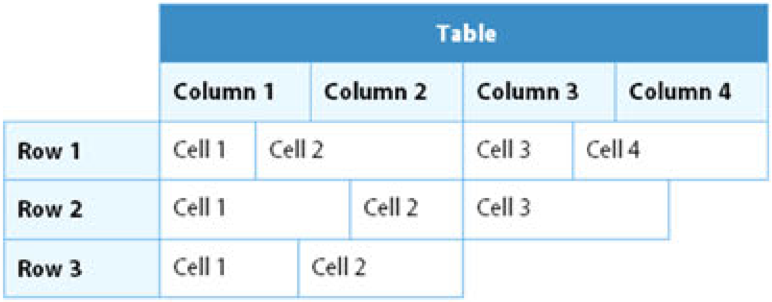


	Table			
	Column 1	Column 2	Column 3	Column 4
Row 1	Cell 1	Cell 2	Cell 3	Cell 4
Row 2	Cell 1	Cell 2	Cell 3	
Row 3	Cell 1	Cell 2		

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.

www.aspose.com

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

EU: +44 141 416 1112
sales.europe@aspose.com

Oceania: +61 2 8003 5926
sales.asiapacific@aspose.com

Case Study: Aspose.Words for .NET

Lulu helps authors, publishers, businesses, and educators publish and sell print on demand books and ebooks. Why do they use Aspose?

LULU IS A TECHNOLOGY COMPANY THAT PROVIDES AN OPEN PUBLISHING PLATFORM where customers from all over the world can create, publish and sell print-on-demand books, ebooks, photobooks and calendars.

The basic function of Lulu's publishing platform is to receive manuscripts from customers and send them to printers for printing. The printers receive the manuscripts in PDF format but that is not always its original format.

> It took only 10 lines of code to integrate Aspose.Words for .NET into the new solution.

Customers can submit manuscripts in any number of formats: many use Microsoft Word. Lulu's publishing platform converts incoming manuscripts to PDFs that can then be sent to the printer. The conversion is automatic: the document comes in, is converted and goes off to print withouthuman intervention or review.

Updating the Platform

Lulu has been running for several years. The original conversion platform depended on Microsoft Automation for converting DOC files to PDFs. As the business grew and had to accommodate a much higher number of manuscripts, some problems with the existing platform became apparent.

- It did not scale,
- it did not support the latest Microsoft Word document formats, and
- it was not robust.

Looking for a Solution

The company decided to build a new platform using components that could support their continued growth.

Lulu's engineering team tested Aspose. Words for .NET alongside the existing Microsoft Automation system and other applications. Each solution had its strengths and weaknesses but in the end, Aspose. Words for .NET won because

- it took only 10 lines of code to integrate it into the new platform,
- it is robust and scalable,
- it supports all the file formats that Lulu needs, and
- the licensing structure is straight-forward and cheaper over time than other solutions.

Outcome

The result was a product that can take any Microsoft Word document that a customer submits, regardless of how the customer may have embellish their manuscript, and convert it to a PDF file that can be printed anywhere.

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

Customers create manuscripts that can be printed by any printer.

Powerful Document Management Solutions for Your Web and Mobile Applications

GroupDocs offers professional stand-alone .NET & Java libraries along with cloud APIs that allow end-users to view, annotate, convert, e-sign, assemble and compare documents and images of more than 35 file formats within your own web and mobile applications. Key benefits include:

- All GroupDocs APIs are 100% independent, and don't require any 3rd party software installation.
- Being extremely lightweight, .NET & Java libraries can be integrated with just a single DLL.
- Cloud APIs are supported by SDKs to help developers on .NET, Java, JavaScript, PHP, Python and Ruby seamlessly integrate GroupDocs solutions into any web or mobile apps.
- No need for client-side installation. End-users can work with documents from any web-enabled device and modern web-browser.
- All GroupDocs' products come with a 30-day fully-functional trial and free support during the integration period.

Stand-Alone .NET & Java Libraries Pricing

GroupDocs .NET & Java licenses are based on the number of developers and the number of locations where the components will be used:

	Developer Small Business	Developer OEM	Site Small Business	Site OEM
License for one developer	✓	✓		
Licenses for up to 10 developers			✓	✓
Use derived work at one location	✓			
Use derived work at up to 10 locations			✓	
Royalty free/deploy to unlimited locations		✓		✓
Discount applied to multiple purchases	✓	✓	✓	✓
Can be used to create unlimited applications	✓	✓	✓	✓
Updates and hotfixes for one year	✓	✓	✓	✓
Free technical support	✓	✓	✓	✓
Price	**$2,499**	**$7,497**	**$9,996**	**$29,988**

Cloud API Pricing

GroupDocs' cloud APIs use a different licensing model. Instead of licenses, they are charged by use: the number of calls made to the API. To find out more about our cloud API pricing, please visit our website: www.groupdocs.com

GroupDocs Viewer

A powerful document viewer API that allows you to display over 35 document formats in your web or mobile application. The viewer can both rasterize documents and convert them to SVG+HTML+CSS, delivering true-text high-fidelity rendering.
Supported file formats include: Microsoft Office, Visio, Project and Outlook documents, PDFs, AutoCAD, image files (TIFF, JPG, BMP, GIF, TIFF, etc.) and more.

GroupDocs Annotation

With support for over 35 file formats, this API allows your app users to annotate documents of all common business formats, including Microsoft Office and PDF. And thanks to the advanced document management options, users can store, share, print, download and export the annotated documents easily - all from within your own application.

GroupDocs Conversion

GroupDocs Conversion API allows end users to convert back and forth between over 25 document formats within your own application. It supports all Microsoft Office document formats as well as PDF, HTML and common image file formats (TIFF, JPEG, GIF, PNG, BMP). Your users can convert documents one by one on the fly, or add several documents at a time to a conversion queue.

GroupDocs Signature

GroupDocs Signature API is an easy way to give your apps legally binding e-signature capabilities. Your users are then able to get documents signed electronically using only a web-browser.
The API gives developers access to sophisticated online signature features, from e-signature capture control and different signing workflows, to reminder management, contact management and signer roles.

GroupDocs Assembly

GroupDocs Assembly automatically incorporates data submitted through online forms into existing document templates in PDF or Microsoft Word formats. For each completed form a new custom document is generated. The API lets you build document assembly solutions without getting into the details of working with templates, fields and merging data.

GroupDocs Comparison

A document comparison API that allows users to quickly and easily find differences between two revisions of a document right in your web or mobile app. It merges two uploaded documents into a single one and displays it, highlighting differences with the redline view approach - similar to the Microsoft Word change tracking feature, but online. Works with Microsoft Word, Excel, and PowerPoint documents, as well as Adobe Acrobat PDF files.

Adding File Conversion and Manipulation to Business Systems

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

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

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

Aspose creates APIs that work independently of Microsoft Office Automation.

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

Document Conversion Options

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

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

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

Look to Aspose

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

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

Finding the Right Tool

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

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

Aspose.BarCode

A complete toolkit for barcode generation and recognition

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

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

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

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

Robust and reliable barcode generation and recognition.

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

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

Common Uses

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

Key Features

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

Supported File Formats

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

Format support varies across platforms.

Supported Barcodes

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

Platforms

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

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

HTM
JPG
XLS
PPT
PDF
MSG

Aspose.Email

Work with emails and calendars without Microsoft Outlook

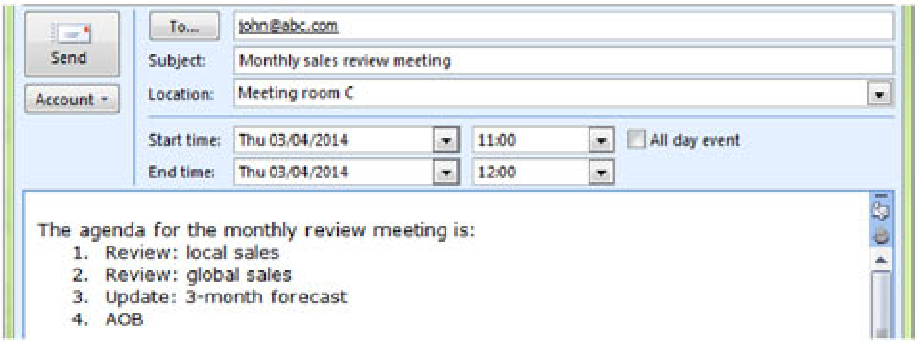
- Complete email processing solution.
- Message file format support.

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.



john@abc.com

Subject: Monthly sales review meeting

Location: Meeting room C

Start time: Thu 03/04/2014 11:00 □ All day event

End time: Thu 03/04/2014 12:00

The agenda for the monthly review meeting is:
1. Review: local sales
2. Review: global sales
3. Update: 3-month forecast
4. AOB

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

Common Uses

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

Key Features

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

Supported File Formats

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

Format support varies across platforms.

Platforms

Pricing Info

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

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

Aspose.Pdf

Create PDF documents without using Adobe Acrobat

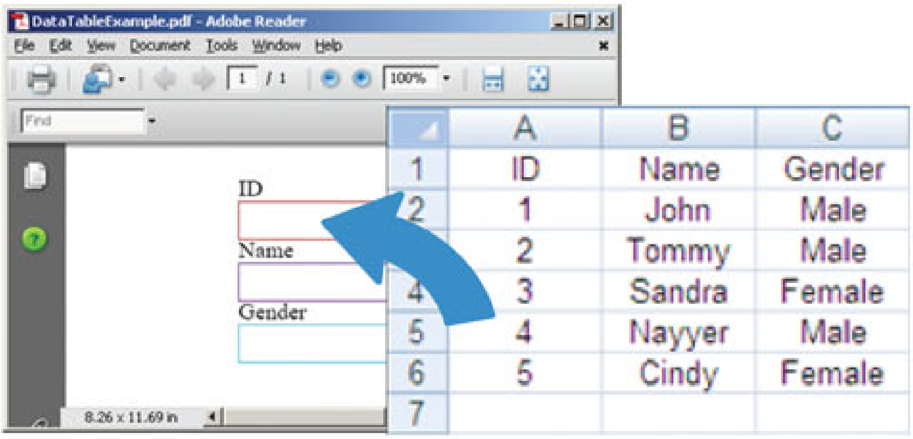
- A complete solution for programming with PDF files.
- Work with PDF forms and form fields.

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

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

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.

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


DataTableExample.pdf - Adobe Reader
File Edit View Document Tools Window Help
1 / 1
100%
Find
ID
Name
Gender
8.26 x 11.69 in

	A	B	C
1	ID	Name	Gender
2	1	John	Male
3	2	Tommy	Male
4	3	Sandra	Female
5	4	Nayyer	Male
6	5	Cindy	Female
7			

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.

www.aspose.com

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

EU: +44 141 416 1112
sales.europe@aspose.com

Oceania: +61 2 8003 5926
sales.asiapacific@aspose.com

Banckle.com - Customer Service 2.0

Customer Service Apps that Help Grow Your Business

Banckle's professional web apps help companies engage with customers.

Banckle Chat

Chat live with your customers to give them the help they want, when they need it. Embeddable HTML makes it easy to integrate Banckle's live chat service into your website.

Prices start from **$6.30**/month.

Banckle Meeting

Give your customers a great online meeting experience with an intuitive interface. Share screen, presentations, work with a whiteboard and use video conferencing from a web browser.

Prices start from **$10.50**/month.

Banckle CRM

Keep the team up to date with projects and customer contacts. Keep an eye on the pipeline, manage tasks and log contacts in one central place to improve team work and customer focus.

Prices start from **$4.20**/month.

Banckle Helpdesk

Stay on top of your customers' issues with an online service desk and ticketing system. Never lose a ticket but let the team collaborate to deliver the best possible customer support.

Prices start from **$11.00**/month.

Banckle Campaign

Find out how effective your email campaigns are and keep in touch with your customers. Design, test and send campaigns, manage mailing lists and learn from campaign reports.

Prices start from **$9.80**/month.

Banckle Email

Manage email from an affordable, secure and intuitive online platform. Use all the email features you are used to: attachments, folders and simple account administration.

Prices start from **$13.30**/month.

Banckle Total

All Banckle's customer service tools in one package, Banckle Total helps you take customer support to the next level.

Subscriptions start at **$31.00**/month.

Try Our APIs

Want to build your own solution?, Consider our RESTful API's

Contact **sales@banckle.com** for API pricing information and an extended 30 day free trial. Alternatively, visit **banckle.com** and use promo code **MiniMag2014** at checkout for a 20% discount.

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.
- OLE integration for embedding external content.
- Wide support for input and output file formats.

Key Features

- A complete presentation development solution.
- Control over text, formatting and slide elements.

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

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.

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.

ASPOSE
Your File Format APIs