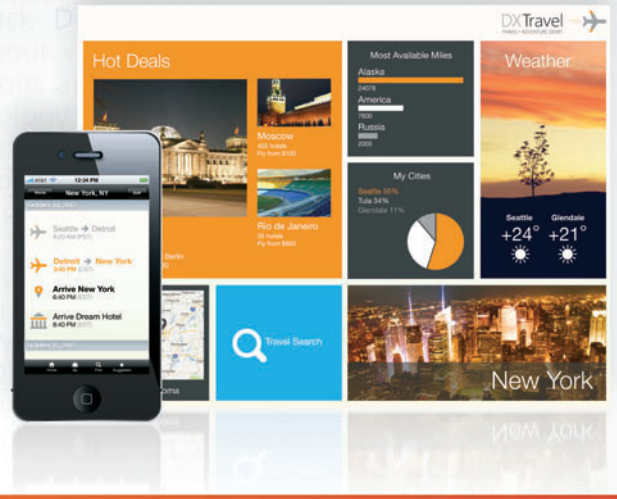
DevExpress

Delight your users by creating apps that feel as though they were designed expressly for the device. With **DXTREME**, multi-channel means building applications that span devices and optimize the best parts of each platform. And with HTML5/JS visualization built in your dynamic charts and graphs will be both powerful and beautiful.



discover DXTREME

Are you ready to go cross-platform?
Download the **DXTREME** Preview to experience the future.

www.DevExpress.com

DXv2

The next generation of inspiring tools. **Today.**

u dxt dxp asp win wpf sl

THE MICROSOFT JOURNAL FOR DEVELOPERS

NOVEMBER 2012 VOL 27 NO 11

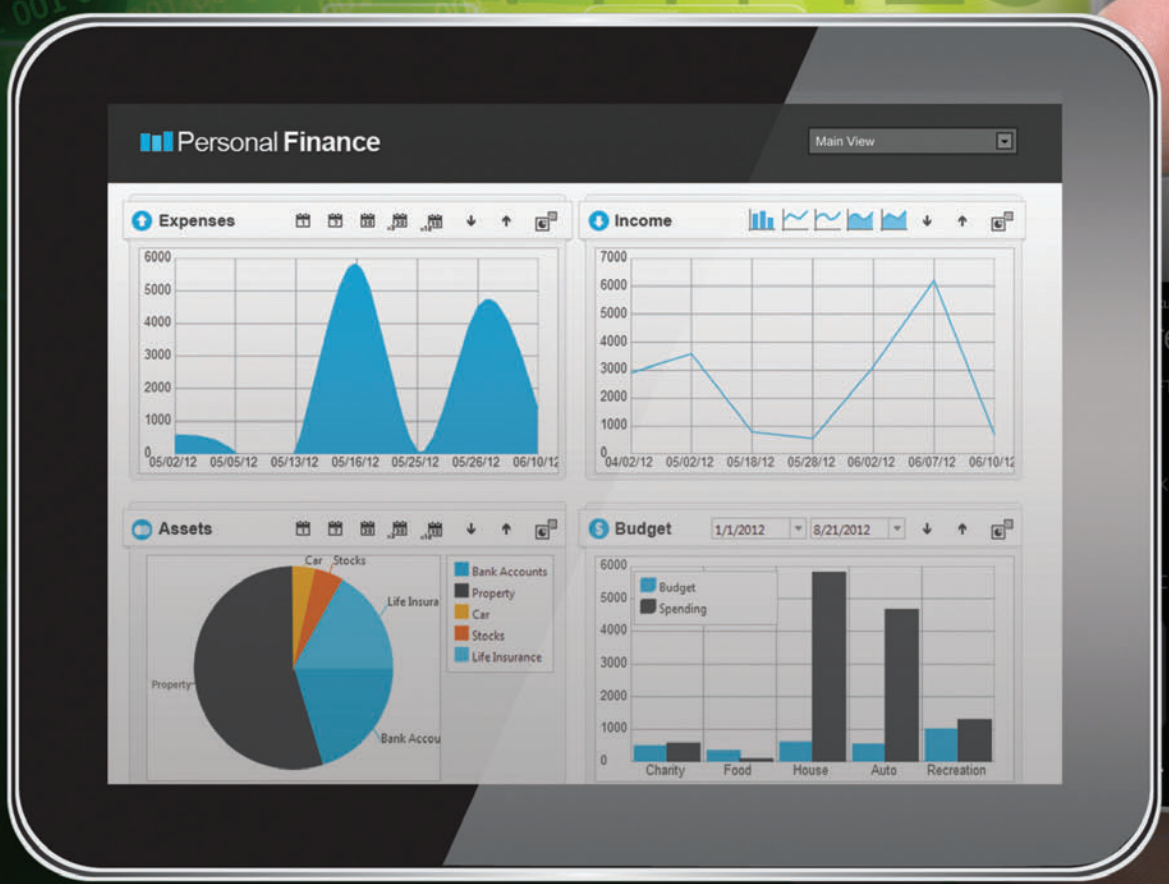
msdn magazine

Windows 8/JavaScript....36, 64

COLUMNS

Microsoft

Compatible with
Microsoft® Visual Studio® 2012



BRILLIANT UX

At Your Fingertips

infragistics.com/ **EXPERIENCE**

INFRAGISTICS™

DESIGN / DEVELOP / EXPERIENCE

Infragistics Sales US 800 231 8588 • Europe +44 (0) 800 298 9055 • India +91 80 4151 8042 • APAC (+61) 3 9982 4545

msdn magazine

NOVEMBER 2012 VOLUME 27 NUMBER 11

MSDN Magazine (ISSN 1528-4859) is published 13 times a year, monthly with a special issue in October 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.

Editor's Note

MICHAEL DESMOND

Standing Up

Back in June I blogged about my decision to switch to a standing desk in my office (bit.ly/Nfuedz). Like many developers, I spend an inordinate amount of time in front of a PC monitor, and the long hours sitting in an office chair were taking a toll. Sore back, stiff neck, aching right shoulder, numb legs—by the end of the day I felt like I had aged 20 years.

What ultimately drove me to my feet was a growing body of evidence that sitting all day is a real health risk. A long-term study of 7,744 men by the University of South Carolina (bit.ly/8XQGQz) found that sedentary lifestyles dramatically increase the risk of dying of cardiovascular disease. Men reporting more than 23 hours of sedentary activity a week were 64 percent more likely to die from heart disease, according to the study, than those reporting less than 11 hours of inactivity. A 2012 study from the University of Missouri, meanwhile, continuously monitored the blood sugar levels of participants over two three-day spans—one featuring regular physical activity and the other reduced activity. Researchers found that participant blood sugar levels spiked during the days of lower activity. (1.usa.gov/nijOZB).

And really, I didn't need medical studies to tell me what my body already knows. Sitting for 10-plus hours a day is not good for me. So I don't sit anymore.

I got started on the cheap. I grabbed a few spare file boxes to serve as pedestals for my two LCD displays (27 and 24 inch), keyboard, mouse and 11.5-inch notebook. The setup had all the visual appeal of a couch dumped on a front lawn, but it worked. And within a couple days I noticed something. The aches I encountered after a long day hunched over the keyboard were gone. Granted, my feet hurt a bit and my knees were sore, but I felt *much* better in the evenings than ever before.

It's been four months since I switched to a standing desk. The file boxes are gone, replaced by a chrome wire shelf set on 14-inch posts to lift my monitors close to eye level. And while I'd love to move to a convertible desk that can be raised or lowered to a sitting or standing position, I'm too cheap to make the commitment. But as one reader pointed out to me in an e-mail exchange, paying $800 or more on a convertible desk, such as the GeekDesk, could be money well spent.

And really, I didn't need medical studies to tell me what my body already knows. Sitting for 10-plus hours a day is not good for me.

"I came to the conclusion that if I can spend $1,000 on ski equipment that gets used maybe 10 to 20 times a year, I can spend about that on a desk that I'll use every day for years," wrote Jason Linden, an analyst/developer at Boeing in Everett, Wash. "You could maybe even deduct it [for tax purposes] if you meet the requirements."

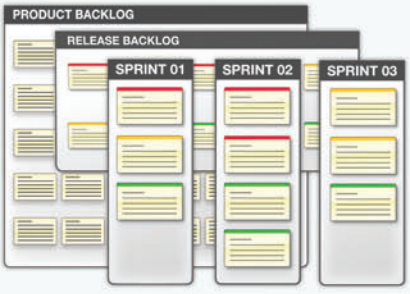
In the meantime, I'm doing what I can to stay comfortable. I take occasional breaks, repairing to a sofa to put my feet up while I read a manuscript or take a quick phone call, and I keep a tall stool handy so I can sit down occasionally at my workstation. I also started standing on a padded yoga mat a couple months back, which has really helped. Another reader wrote that a small foot stool under the desk can help with fatigue as well, and I plan to give that a try.

I'll continue to tweak things, but after four months it's clear that my standing desk experiment has been a huge success. Have you moved to a standing desk or found another solution? E-mail me at mmeditor@microsoft.com.

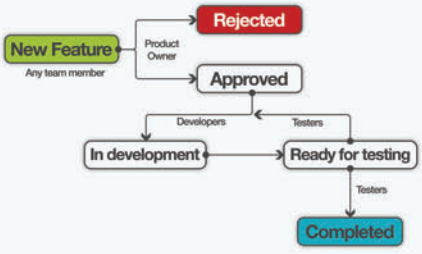
OnTime Scrum

Agile project management & bug tracking software

The **Scrum project management** tool your development team will **love to use.**



Easily manage product backlogs



Automate your workflow process

Project visibility with burndowns

Enterprise-quality software at prices **any team can afford.**

$10 per month for up to 10 users

special small-team pricing

$7 per user per month

for teams of 11+ users

Exclusive offer for MSDN readers: **3 months free** of **OnTime Scrum**

Visit **OnTimeNow.com/MSDN**

OnTime Scrum offers your dev team:

- flexible product backlog management
- powerful user story & bug tracking
- highly configurable user roles & workflows
- automated burndown charts
- **new** real-time visual project dashboard
- Visual Studio & Github integration
- all in a fast, versatile HTML5 interface

axosoft

800.653.0024 • www.ontimenow.com • www.axosoft.com • @axosoft

KENNY KERR

The Evolution of Synchronization in Windows and C++

When I first started writing concurrent software, C++ had no support for synchronization. Windows itself had only a handful of synchronization primitives, all of which were implemented in the kernel. I tended to use a critical section unless I needed to synchronize across processes, in which case I used a mutex. In general terms, both of these are locks, or lock objects.

The mutex takes its name from the concept of "mutual exclusion," another name for synchronization. It refers to the guarantee that only one thread can access some resource at a time. The critical section takes its name from the actual section of code that might be accessing such a resource. To ensure correctness, only one thread can execute this critical section of code at a time. These two lock objects have different capabilities, but it's helpful to remember that they're just locks, they both provide mutual exclusion guarantees and both can be used to demarcate critical sections of code.

Figure 1 **The Critical Section Lock**

```
class lock
{
  CRITICAL_SECTION h;

  lock(lock const &);
  lock const & operator=(lock const &);

public:

  lock()
  {
    InitializeCriticalSection(&h);
  }

  ~lock()
  {
    DeleteCriticalSection(&h);
  }

  void enter()
  {
    EnterCriticalSection(&h);
  }

  bool try_enter()
  {
    return 0 != TryEnterCriticalSection(&h);
  }

  void exit()
  {
    LeaveCriticalSection(&h);
  }

  CRITICAL_SECTION * handle()
  {
    return &h;
  }
};
```

Today the synchronization landscape has changed dramatically. There are a plethora of choices for the C++ programmer. Windows now supports many more synchronization functions, and C++ itself at long last provides an interesting collection of concurrency and synchronization capabilities for those using a compiler supporting the C++11 standard.

In this month's column I'm going to explore the state of synchronization in Windows and C++. I'll start with a review of the synchronization primitives provided by Windows itself and then consider the alternatives provided by the Standard C++ Library. If portability is your main concern, then the new C++ library additions will be very appealing. If, however, portability is less of a concern and performance is paramount, then getting familiar with what Windows now offers will be important. Let's dive right in.

Critical Section

First up is the critical section object. This lock is used heavily by countless applications but has a sordid history. When I first started using critical sections, they were really simple. To create such a lock, all you needed was to allocate a CRITICAL_SECTION structure and call the InitializeCriticalSection function to prepare it for use. This function doesn't return a value, implying that it can't fail. Back in those days, however, it was necessary for this function to create various system resources, notably a kernel event object, and it was possible that in extremely low-memory situations this would fail, resulting in a structured exception being raised. Still, this was rather rare, so most developers ignored this possibility.

With the popularity of COM, the use of critical sections skyrocketed because many COM classes used critical sections for synchronization, but in many cases there was little to no actual contention to speak of. When multiprocessor computers became more widespread, the critical section's internal event saw even less usage because the critical section would briefly spin in user mode while waiting to acquire the lock. A small spin count meant that many short-lived periods of contention could avoid a kernel transition, greatly improving performance.

Around this time some kernel developers realized that they could dramatically improve the scalability of Windows if they

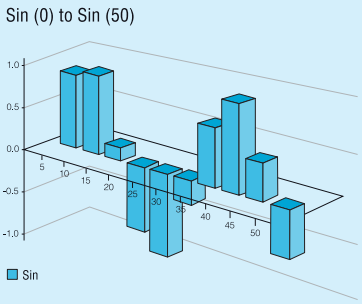
ALTOVA® missionkit®

Create cutting edge reports with the complete set of tools from Altova®

Experience how Altova MissionKit®, the integrated suite of XML, SQL, and data integration tools, lets you display and analyze data through enhanced chart and report generation.

NEW in Version 2013:

- Watermark support
- Conditions for table rows & columns
- Smart Fix XML validation with automatic error correction
- Support for SQL stored procedures in data mapping



Report generation is finally easy – and affordable – with Altova MissionKit reporting tools:

StyleVision® – stylesheet and report design tool

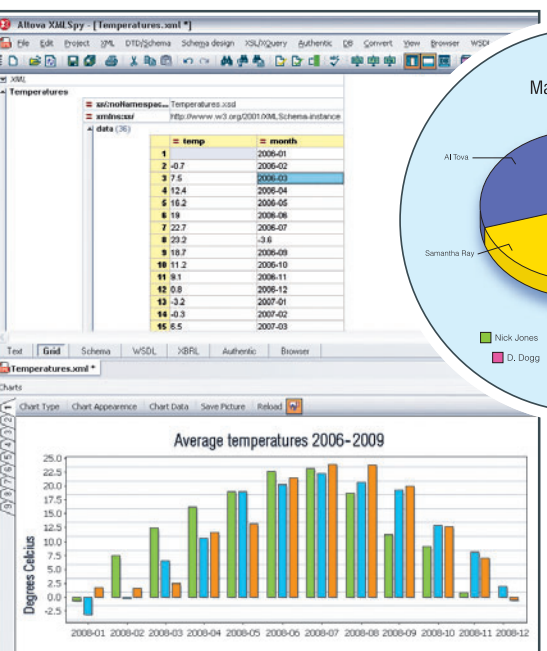
- Consolidated reporting based on XML, SQL database, and/or XBRL data
- Dynamic data selection and rendering
- HTML, Microsoft® Word, PDF, and e-Form report creation

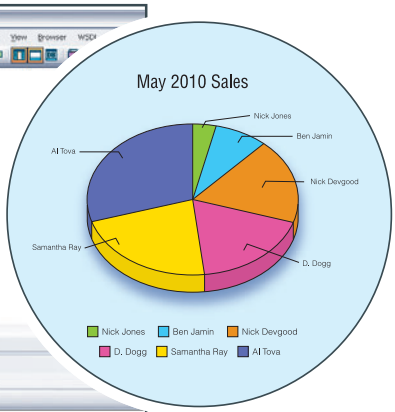
XMLSpy® – advanced XML editor

- Instant creation of charts to analyze XML data
- One-click export of charts to XSLT, XQuery, or image files

MapForce® – any-to-any data mapping tool

- Integration with StyleVision for rendering attractive reports
- Report generation for virtually any data format: XML, databases, EDI, flat files, Excel, and more





Download a 30 day free trial!

Try before you buy with a free, fully functional, trial from **www.altova.com**

Scan to learn more about MissionKit reporting tools.

deferred the creation of critical section event objects until there was enough contention to necessitate their presence. This seemed like a good idea until the developers realized this meant that although InitializeCriticalSection could now not possibly fail, the EnterCriticalSection function (used to wait for lock ownership) was no longer reliable. This could not as easily be overlooked by developers, because it introduced a variety of failure conditions that would've made critical sections all but impossible to use correctly and break countless applications. Still, the scalability wins couldn't be overlooked.

A kernel developer finally arrived at a solution in the form of a new, and undocumented, kernel event object called a keyed event. You can read a little about it in the book, "Windows Internals," by Mark E. Russinovich, David A. Solomon and Alex Ionescu (Microsoft Press, 2012), but basically, instead of requiring an event object for every critical section, a single keyed event could be used for all critical sections in the system. This works because a keyed event object is just that: It relies on a key, which is just a pointer-sized identifier that's naturally address-space local.

Figure 2 **The Mutex Lock**

```
#ifdef _DEBUG
  #include <crtdbg.h>
  #define ASSERT(expression) _ASSERTE(expression)
  #define VERIFY(expression) ASSERT(expression)
  #define VERIFY_(expected, expression) ASSERT(expected == expression)
#else
  #define ASSERT(expression) ((void)0)
  #define VERIFY(expression) (expression)
  #define VERIFY_(expected, expression) (expression)
#endif

class lock
{
  HANDLE h;

  lock(lock const &);
  lock const & operator=(lock const &);

public:

  lock() :
    h(CreateMutex(nullptr, false, nullptr))
  {
    ASSERT(h);
  }

  ~lock()
  {
    VERIFY(CloseHandle(h));
  }

  void enter()
  {
    VERIFY_(WAIT_OBJECT_0, WaitForSingleObject(h, INFINITE));
  }

  bool try_enter()
  {
    return WAIT_OBJECT_0 == WaitForSingleObject(h, 0);
   }

  void exit()
  {
    VERIFY(ReleaseMutex(h));
  }

  HANDLE handle()
  {
    return h;
  }
};
```

There was surely a temptation to update critical sections to use keyed events exclusively, but because many debuggers and other tools rely on the internals of critical sections, the keyed event was only used as a last resort if the kernel failed to allocate a regular event object.

This may sound like a lot of irrelevant history but for the fact that the performance of keyed events was significantly improved during the Windows Vista development cycle, and this lead to the introduction of a completely new lock object that was both simpler and faster—but more on that in a minute.

As the critical section object is now exempt from failure due to low-memory conditions, it really is very straightforward to use. **Figure 1** provides a simple wrapper.

The EnterCriticalSection function I already mentioned is complemented by a TryEnterCriticalSection function that provides a nonblocking alternative. The LeaveCriticalSection function releases the lock, and DeleteCriticalSection releases any kernel resources that might have been allocated along the way.

So the critical section is a reasonable choice. It performs quite well as it attempts to avoid kernel transitions and resource allocation. Still, it has a bit of baggage that it must carry due to its history and application compatibility.

Mutex

The mutex object is a true kernel-synchronization object. Unlike critical sections, a mutex lock always consumes kernel-allocated resources. The benefit, of course, is that the kernel is then able to provide cross-process synchronization due to its awareness of the lock. As a kernel object, it provides the usual attributes—such as a

Figure 3 **The Event Signal**

```
class event
{
  HANDLE h;

  event(event const &);
  event const & operator=(event const &);

public:

  explicit event(bool manual = false) :
    h(CreateEvent(nullptr, manual, false, nullptr))
  {
    ASSERT(h);
  }

  ~event()
  {
    VERIFY(CloseHandle(h));
  }

  void set()
  {
    VERIFY(SetEvent(h));
  }

  void clear()
  {
    VERIFY(ResetEvent(h));
  }

  void wait()
  {
    VERIFY_(WAIT_OBJECT_0, WaitForSingleObject(h, INFINITE));
  }
};
```

Review Assistant

Code review is dull and tedious?

Make it fast and easy!

Review Assistant

Code review add-in for Visual Studio.

devart
WWW.DEVART.COM

Figure 4 **The SRW Lock**

```
class lock
{
  SRWLOCK h;

  lock(lock const &);
  lock const & operator=(lock const &);

public:

  lock()
  {
    InitializeSRWLock(&h);
  }

  void enter()
  {
    AcquireSRWLockExclusive(&h);
  }

  bool try_enter()
  {
    return 0 != TryAcquireSRWLockExclusive(&h);
  }

  void exit()
  {
    ReleaseSRWLockExclusive(&h);
  }

  SRWLOCK * handle()
  {
    return &h;
  }
};
```

name—that can be used to open the object from other processes or just to identify the lock in a debugger. You can also specify an access mask to restrict access to the object. As an intraprocess lock, it's overkill, a little more complicated to use and a lot slower. **Figure 2** provides a simple wrapper for an unnamed mutex that's effectively process-local.

The CreateMutex function creates the lock, and the common CloseHandle function closes the process handle, which effectively decrements the lock's reference count in the kernel. Waiting for lock ownership is accomplished with the general-purpose WaitForSingleObject function, which checks and optionally waits for the signaled state of a variety of kernel objects. Its second parameter indicates how long the calling thread should block while waiting to acquire the lock. The INFINITE constant is—not surprisingly—an indefinite wait, while a value of zero prevents the thread from waiting at all and will only acquire the lock if it's free. Finally, the ReleaseMutex function releases the lock.

The mutex lock is a big hammer with a lot of power, but it comes at a cost to performance and complexity. The wrapper in **Figure 2** is littered with assertions to indicate the possible ways that it can fail, but it's the performance implications that disqualify the mutex lock in most cases.

Event

Before I talk about a high-performance lock, I need to introduce one more kernel-synchronization object, one to which I alluded already. Although not actually a lock, in that it doesn't provide a facility to directly implement mutual exclusion, the event object is vitally important for the coordination of work among threads. In fact, it's the same object used internally by the critical section lock—and besides, it comes in handy when implementing all kinds of concurrency patterns in an efficient and scalable way.

The CreateEvent function creates the event and—like the mutex—the CloseHandle function closes the handle, releasing the object in the kernel. Because it isn't actually a lock, it has no acquire/release semantics. Rather, it's the very embodiment of the signaling functionality provided by many of the kernel objects. To understand how the signaling works, you need to appreciate that an event object can be created in one of two states. If you pass TRUE to CreateEvent's second parameter, then the resulting event object is said to be a manual-reset event; otherwise an auto-reset event is created. A manual-reset event requires that you manually set and reset the object's signaled state. The SetEvent and ResetEvent functions are provided for this purpose. An auto-reset event automatically resets (changes from signaled to nonsignaled) when a waiting thread is released. So an auto-reset event is useful when one thread needs to coordinate with one other thread, whereas a manual-reset event is useful when one thread needs to coordinate with any number of threads. Calling SetEvent on an auto-reset event will release at most one thread, whereas with a manual-reset event, that call will release all waiting threads. Like the mutex, waiting for an event to become signaled is achieved with the WaitForSingleObject function. **Figure 3** provides a simple wrapper for an unnamed event that can be constructed in either mode.

Slim Reader/Writer Lock

The name of the Slim Reader/Writer (SRW) lock might be a mouthful, but the operative word is "slim." Programmers might overlook this lock because of its ability to distinguish between shared readers

Figure 5 **The Condition Variable**

```
class condition_variable
{
  CONDITION_VARIABLE h;

  condition_variable(condition_variable const &);
  condition_variable const & operator=(condition_variable const &);

public:

  condition_variable()
  {
    InitializeConditionVariable(&h);
  }

  template <typename T>
  void wait_while(lock & x, T predicate)
  {
    while (predicate())
    {
      VERIFY(SleepConditionVariableSRW(&h, x.handle(), INFINITE, 0));
    }
  }

  void wake_one()
  {
    WakeConditionVariable(&h);
  }

  void wake_all()
  {
    WakeAllConditionVariable(&h);
  }
};
```

componentart.com/windows8

and exclusive writers, perhaps thinking that this is overkill when all they need is a critical section. As it turns out, this is the simplest lock to deal with and also by far the fastest, and you certainly don't need to have shared readers in order to use it. It has this speedy reputation not only because it relies on the efficient keyed event object, but also because it's mostly implemented in user mode and only falls back to the kernel if contention is such that the thread would be better off sleeping. Again, the critical section and mutex objects provide additional features you might require, such as recursive or inter-process locks, but more often than not, all you need is a fast and lightweight lock for internal use.

This lock relies exclusively on the keyed events I mentioned before, and as such it's extremely lightweight despite providing a great deal of functionality. The SRW lock requires only a pointer-sized amount of storage, which is allocated by the calling process rather than the kernel. For this reason, the initialization function, InitializeSRWLock, can't fail and merely ensures the lock contains the appropriate bit pattern before being used.

Waiting for lock ownership is achieved using either the AcquireSRWLockExclusive function for a so-called writer lock or using the AcquireSRWLockShared function for reader locks. However, the exclusive and shared terminology is more appropriate. There are corresponding release and try-acquire functions as one would expect for both exclusive and shared modes. **Figure 4** provides a simple wrapper for an exclusive-mode SRW lock. It wouldn't be hard for you to add the shared-mode functions if needed. Notice, however, that there's no destructor because there are no resources to free.

Figure 6 **Auto-Reset Blocking Queue**

```
template <typename T>
class blocking_queue
{
  std::deque<T> q;
  lock x;
  event e;

  blocking_queue(blocking_queue const &);
  blocking_queue const & operator=(blocking_queue const &);

public:

  blocking_queue()
  {
  }

  void push(T const & value)
  {
    lock_block block(x);
    q.push_back(value);
    e.set();
  }

  T pop()
  {
    lock_block block(x);

    while (q.empty())
    {
      x.exit(); e.wait(); // Bug!
      x.enter();
    }

    T v = q.front();
    q.pop_front();
    return v;
  }
};
```

Figure 7 **Manual-Reset Blocking Queue**

```
template <typename T>
class blocking_queue
{
  std::deque<T> q;
  lock x;
  event e;

  blocking_queue(blocking_queue const &);
  blocking_queue const & operator=(blocking_queue const &);

public:

  blocking_queue() :
    e(true) // manual
  {
  }

  void push(T const & value)
  {
    lock_block block(x);
    q.push_back(value);

    if (1 == q.size())
    {
      e.set();
    }
  }

  T pop()
  {
    lock_block block(x);

    while (q.empty())
    {
      x.exit();
      e.wait();
      x.enter();
    }

    T v = q.front();
    q.pop_front();

    if (q.empty())
    {
      e.clear();
    }

    return v;
  }
};
```

Condition Variable

The final synchronization object I need to introduce is the condition variable. This is perhaps the one with which most programmers will be unfamiliar. I have, however, noticed a renewed interest in condition variables in recent months. This might have something to do with C++11, but the idea isn't new and support for this concept has been around for some time on Windows. Indeed, the Microsoft .NET Framework has supported the condition variable pattern since the very first release, although it was merged into the Monitor class, limiting its usefulness in some ways. But this renewed interest is also thanks to the amazing keyed events that allowed condition variables to be introduced by Windows Vista, and they have only improved since. Although a condition variable is merely a concurrency pattern and, thus, can be implemented with other primitives, its inclusion in the OS means that it can achieve amazing performance and frees the programmer from having to ensure the correctness of such code. Indeed, if you're employing OS synchronization primitives, it's almost impossible to ensure the correctness of some concurrency patterns without the help of the OS itself.

componentart.com/windows8

The condition variable pattern is quite common if you think about it. A program needs to wait for some condition to be met before it can proceed. Evaluating this condition involves acquiring a lock to evaluate some shared state. If, however, the condition hasn't yet been met, then the lock must be released to allow some other thread to fulfill the condition. The evaluating thread must then wait until such time as the condition is met before once again acquiring the lock. Once the lock is reacquired, the condition must again be evaluated to avoid the obvious race condition. Implementing this is harder than it seems because there are, in fact, other pitfalls to worry about—and implementing it in an efficient way is harder still. The following pseudocode illustrates the problem:

```
lock-enter

while (!condition-eval)
{
  lock-exit
  condition-wait
  lock-enter
}

// Do interesting stuff here

lock-exit
```

But even in this illustration there lies a subtle bug. In order to function correctly, the condition must be waited upon before the lock is exited, but doing so wouldn't work because the lock would then never be released. The ability to atomically release one object and wait on another is so critical that Windows provides the SignalObjectAndWait function to do so for certain kernel objects. But because the SRW lock lives mostly in user mode, a different solution is needed. Enter condition variables.

Figure 8 **Condition Variable Blocking Queue**

```
template <typename T>
class blocking_queue
{
  std::deque<T> q;
  lock x;
  condition_variable cv;

  blocking_queue(blocking_queue const &);
  blocking_queue const & operator=(blocking_queue const &);

public:

  blocking_queue()
  {
  }

  void push(T const & value)
  {
    lock_block block(x);
    q.push_back(value);
    cv.wake_one();
  }

  T pop()
  {
    lock_block block(x);

    cv.wait_while(x, [&]()
    {
      return q.empty();
    });

    T v = q.front();
    q.pop_front();
    return v;
  }
};
```

Like the SRW lock, the condition variable occupies only a single pointer-sized amount of storage and is initialized with the fail-safe InitializeConditionVariable function. As with SRW locks, there are no resources to free, so when the condition variable is no longer required the memory can simply be reclaimed.

Because the condition itself is program-specific, it's left to the caller to write the pattern as a while loop with the body being a single call to the SleepConditionVariableSRW function. This function atomically releases the SRW lock while waiting to be woken up once the condition is met. There's also a corresponding SleepConditionVariableCS function if you wish to use condition variables with a critical section lock instead.

The WakeConditionVariable function is called to wake a single waiting, or sleeping, thread. The woken thread will reacquire the lock before returning. Alternatively, the WakeAllConditionVariable function can be used to wake all waiting threads. **Figure 5** provides a simple wrapper with the necessary while loop. Note that it's possible for the sleeping thread to be woken unpredictably, and the while loop ensures that the condition is always rechecked after the thread reacquires the lock. It's also important to note that the predicate is always evaluated while holding the lock.

Blocking Queue

To give this some shape, I'll use a blocking queue as an example. Let me stress that I do not recommend blocking queues in general. You may be better served using an I/O completion port or the Windows thread pool, which is just an abstraction over the former, or even the Concurrency Runtime's concurrent_queue class.

Figure 9 **C++11 Blocking Queue**

```
template <typename T>
class blocking_queue
{
  std::deque<T> q;
  std::mutex x;
  std::condition_variable cv;

  blocking_queue(blocking_queue const &);
  blocking_queue const & operator=(blocking_queue const &);

public:

  blocking_queue()
  {
  }

  void push(T const & value)
  {
    std::lock_guard<std::mutex> lock(x);
    q.push_back(value);
    cv.notify_one();
  }

  T pop()
  {
    std::unique_lock<std::mutex> lock(x);

    cv.wait(lock, [&]()
    {
      return !q.empty();
    });

    T v = q.front();
    q.pop_front();
    return v;
  }
};
```

Anything nonblocking is generally preferred. Still, a blocking queue is a simple concept to grasp and something many developers seem to find useful. Admittedly, not every program needs to scale, but every program needs to be correct. A blocking queue also provides ample opportunity to employ synchronization for correctness—and, of course, ample opportunity to get it wrong.

Consider implementing a blocking queue with just a lock and an event. The lock protects the shared queue and the event signals to a consumer that a producer has pushed something onto the queue. **Figure 6** provides a simple example using an auto-reset event. I used this event mode because the push method queues only a single element and, thus, I only want one consumer to be woken to pop it off the queue. The push method acquires the lock, queues the element and then signals the event to wake up any waiting consumer. The pop method acquires the lock and then waits until the queue isn't empty before dequeuing an element and returning it. Both methods use a lock_block class. For brevity it hasn't been included, but it simply calls the lock's enter method in its constructor and the exit method in its destructor.

However, notice the likely deadlock because the exit and wait calls aren't atomic. If the lock were a mutex, I could use the SignalObjectAndWait function, but the performance of the blocking queue would suffer.

Another option is to use a manual-reset event. Rather than signaling whenever an element is queued, simply define two states. The event can be signaled for as long as there are elements in the queue and nonsignaled when it's empty. This will also perform a lot better because there are fewer calls into the kernel to signal the event. **Figure 7** provides an example of this. Notice how the push method sets the event if the queue has one element. This avoids unnecessary calls to the SetEvent function. The pop method dutifully clears the event if it finds the queue empty. As long as there are multiple queued elements, any number of consumers can pop elements off the queue without involving the event object, thus improving scalability.

In this case there's no potential deadlock in the exit-wait-enter sequence because another consumer can't steal the event, given that it's a manual-reset event. It's hard to beat that in terms of performance. Nevertheless, an alternative (and perhaps more natural) solution is to use a condition variable instead of an event. This is easily done with the condition_variable class in **Figure 5** and is similar to the manual-reset blocking queue, although it's a bit simpler. **Figure 8** provides an example. Notice how the semantics and concurrency intentions become clearer as the higher-level synchronization objects are employed. This clarity helps to avoid concurrency bugs that often plague more-obscure code.

Finally, I should mention that C++11 now provides a lock, called a mutex, as well as a condition_variable. The C++11 mutex has nothing to do with the Windows mutex. Likewise, the C++11 condition_variable isn't based on the Windows condition variable. This is good news in terms of portability. It can be used anywhere you can find a conforming C++ compiler. On the other hand, the C++11 implementation in the Visual C++ 2012 release performs quite poorly compared to the Windows SRW lock and condition variable. **Figure 9** provides an example of a blocking queue implemented with the Standard C++11 Library types.

The Standard C++ Library implementation will undoubtedly improve in time, as will the library's support for concurrency in general. The C++ committee has taken some small, conservative steps toward concurrency support that should be recognized, but the work is not yet done. As I discussed in my last three columns, the future of C++ concurrency is still in question. For now, the combination of some excellent synchronization primitives in Windows and the state-of-the-art C++ compiler makes for a compelling toolkit for producing lightweight and scalable concurrency-safe programs. ■

Kenny Kerr *is a software craftsman with a passion for native Windows development. Reach him at kennykerr.ca.*

Thanks to the following technical expert for reviewing this article: *Mohamed Ameen Ibrahim*

JULIE LERMAN

Entity Framework Designer Gets Some Love in Visual Studio 2012

I have to admit, I've been quite enamored with using Code First to build domain models that Entity Framework can use. And I've not paid a lot of attention to EDMX models, which you can build using the Entity Framework Designer via either Database First or Model First models. (See my May 2011 column, "Demystifying Entity Framework Strategies: Model Creation Workflow" [msdn.microsoft.com/magazine/hh148150], to learn about the differences between Code First, Model First and Database First.)

But way back when, I lived in the designer. Ready for the "I'm so old" jokes? I'm so old I used Entity Framework when you had to work with raw XML because there was no designer. It's true. This was during the early betas before the first release. So the designer that came in Visual Studio 2008 was a major blessing. Microsoft did a great job, even though it's always easy to find ways it could be better … or complain about what's not there. In Visual Studio 2010, we gained Model First support and some other nice features in the designer such as pluralization and foreign key support.

Now, Visual Studio 2012 brings an even more enhanced designer. There are two big changes but also a lot of minor tweaks that just make life easier and bring a smile to my face—hopefully to yours, too.

1, 2, 3, Color!

You may already have heard about one notable improvement—the ability to add color to entities in the designer. And it's a pretty simple feature to use. You can select one or more entities and then, in the properties window, choose a color for those entities as shown in **Figure 1**. Color is great for tagging entities visually. When you open a model, it's very easy to see that certain entities relate to one another within the model. Even in a small model such as the one in **Figure 2**, it's clear how useful this can be.

Organize with Diagrams

The debates rage on about how many entities are too many for a model. For my brain, more than about 20 or 30 entities in a single model are visually and physically unmanageable. I mean that it's both visually hard to take in all of that information at once and, when I want to work on one or more entities in the designer, it can be cumbersome. I often resort to the designer's Model Browser to help me locate an entity on the design surface. With huge models (hundreds and hundreds of entities—something I just won't ever attempt), you might experience design-time and runtime performance issues. (See the Microsoft Data Developer Center topic, "Performance Considerations for Entity Framework 5," at bit.ly/OZTiUL.) There are third-party EDMX designers that overcome this problem by taking a different approach to working with large models. My preference is to create small models for use throughout my apps, but that's not always an option. Having the ability to manage many entities in a single model in the designer is important.

The new designer helps developers with this problem by providing an oft-requested feature—diagrams. With model diagrams, you can create different design-time views of what's in the model. You might want to have one view that displays the entire model, such as the one in **Figure 2**, and then use additional diagrams to represent different ways of grouping the entities. The designer provides a number of approaches for creating diagrams, but be careful of functions that will move an entity rather than duplicate it.

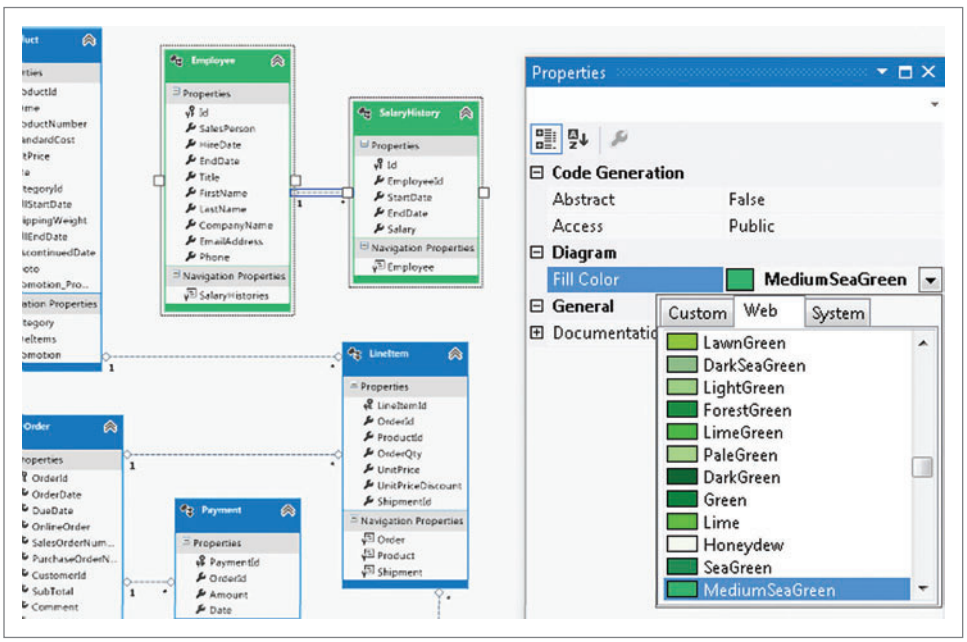
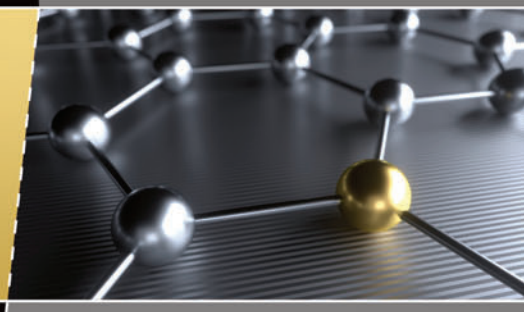


Figure 1 **Changing the Color of Entities**

Powerful Tools for Developers

New! V4.6



Create & Edit PDFs in .Net - ActiveX - WinRT

- Edit, process and print PDF 1.7 documents programmatically.
- Fast and lightweight 32 and 64-bit components for .Net, COM and WinRT applications.
- Create, fill-out and annotate PDF forms.

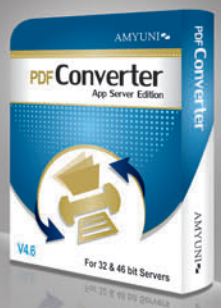
Complete Suite of Accurate PDF Components

- 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, Word, Excel or Silverlight.

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 Driver



- 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 2008 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.

Windows 7 | CERTIFIED FOR Windows Server 2008 | Ready for Windows 8

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

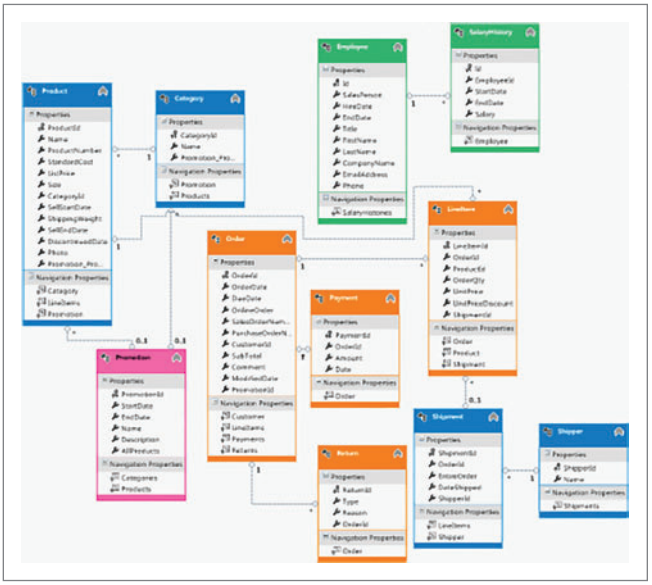


Figure 2 **Easily Visualize Entity Groupings with Color**

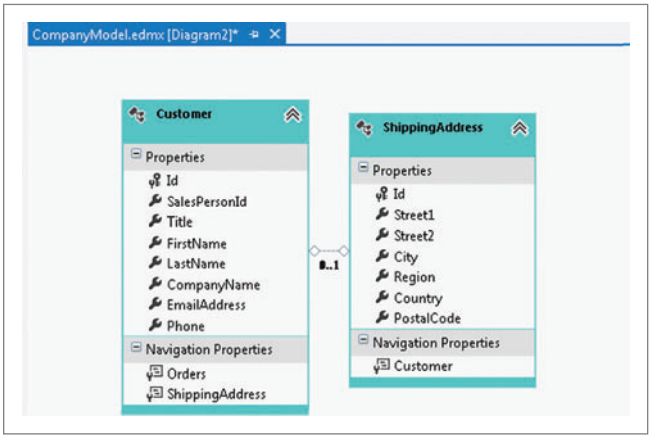


Figure 3 **Diagram Created Using the Move to New Diagram Feature**

For example, I may want a diagram that presents only the entities I'll need when focusing on customer management at design time. That might be Customer and ShippingAddresses in this particular model. There are a few ways you can achieve this. One is by selecting those entities, right-clicking on them and choosing the Move to New Diagram option from the context menu. This will create a new diagram with only those two entities, as shown in **Figure 3**. But because I used the Move option, those two entities are removed from the main diagram. You have to be careful if you want to keep a single diagram of the overall model. I like that the diagram is not a reflection of the model schema in that you can see navigation properties (for example, Customer.Orders) even though in that particular diagram the Orders entity isn't visible.

You can also create and manage diagrams through the designer's Model Browser, shown in **Figure 4**. From the Model Browser you can select the diagram to view, as well as add and delete diagrams. You can also drag entities from the model (for example, in the Entity Types section of the CompanyDatabaseModel) onto a diagram surface. So if I wanted Customer and ShippingAddress to remain on Diagram 1, I could create a new diagram from the Model Browser and then drag Customer and ShippingAddress onto the surface.

Additionally, you can copy and paste entities from one diagram to another and any relevant associations will come along as well.

Keep in mind that the diagrams are design-time views—they don't define the model in its entirety. You can always see the complete model in the Model Browser. Therefore, the duplication of entities from one diagram to another has no effect on your model or your application. But they're not read-only views. You can still design your model and entities in these diagrams and those changes will affect the underlying model. I also appreciate that with any entity in a diagram, you can right-click on that entity and choose the "Include Related" option to pull in all entities that are directly related.

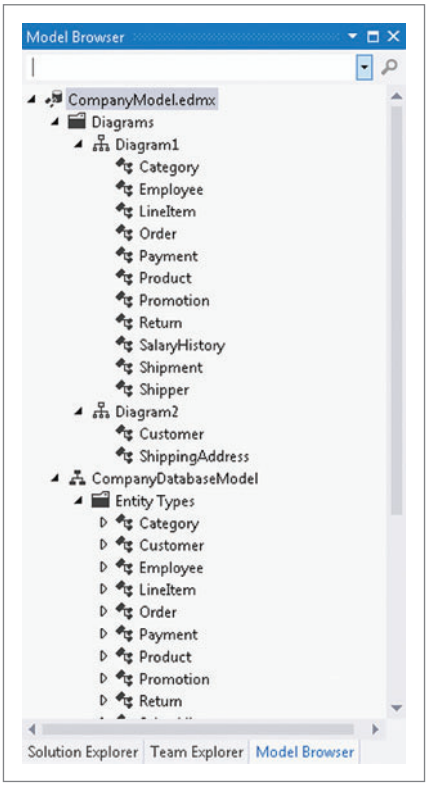


Figure 4 **Manage Diagrams from the Model Browser**

Default Code Generation: POCOs and DbContext

Another change that made me smile when I first discovered it is that, by default, the wizard for creating a new Entity Data Model now uses the Entity Framework 5 DbContext T4 Template. Since the release of Entity Framework 4.1 (which included Code First and the DbContext API), the EF team has recommended that developers begin new projects with the DbContext and Plain Old CLR Object (POCO) classes instead of the .NET 4 ObjectContext and classes that inherit from EntityObject. But Visual Studio 2010 used a code-generation template that defaulted to ObjectContext and EntityObjects. It was up to developers to install the EntityFramework.dll NuGet package and switch their EDMX to use the DbContext template. This meant you had to be aware of the new guidance and act on it.

Now the designer defaults to using the EF 5.x DbContext Generator template, creating a DbContext class and POCOs that are so much easier to use in your apps than the earlier types. The wizard will also pull in the reference to EntityFramework.dll, which is installed onto your computer along with Visual Studio 2012, so you don't even need an Internet connection.

You can easily revert to using ObjectContext. Check the Microsoft Data Developer

The #1 Native Toolset for Building Windows 8 Apps

Build for the consumer and enterprise markets faster with either XAML or HTML.

XAML HTML

www.telerik.com/win8

telerik
deliver more than expected

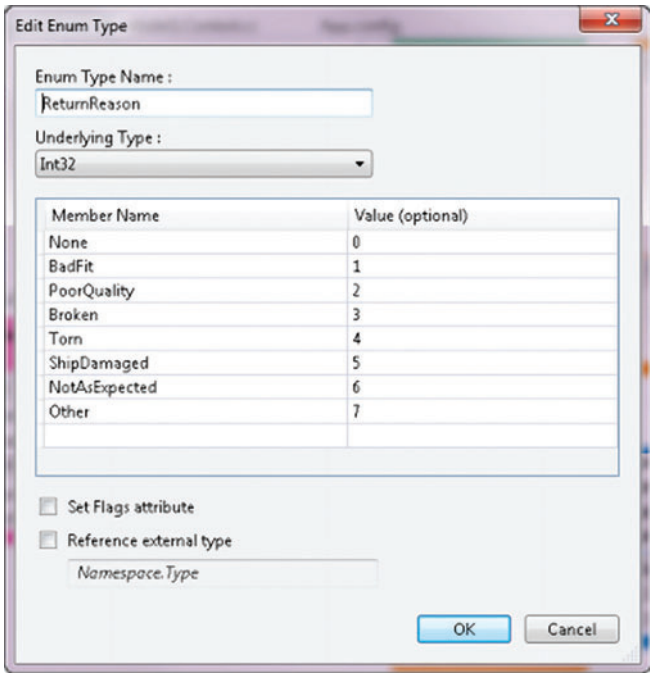


Figure 5 **Defining an Enum Type**

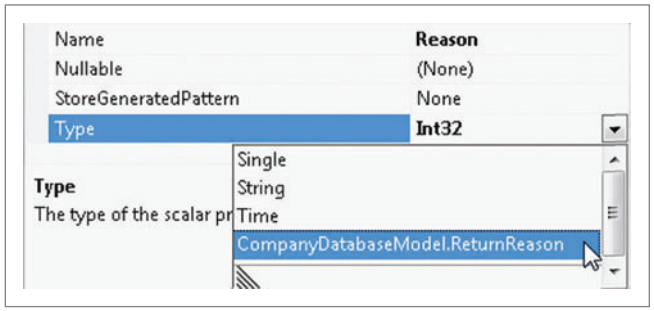


Figure 6 **Selecting a New Enum Type**

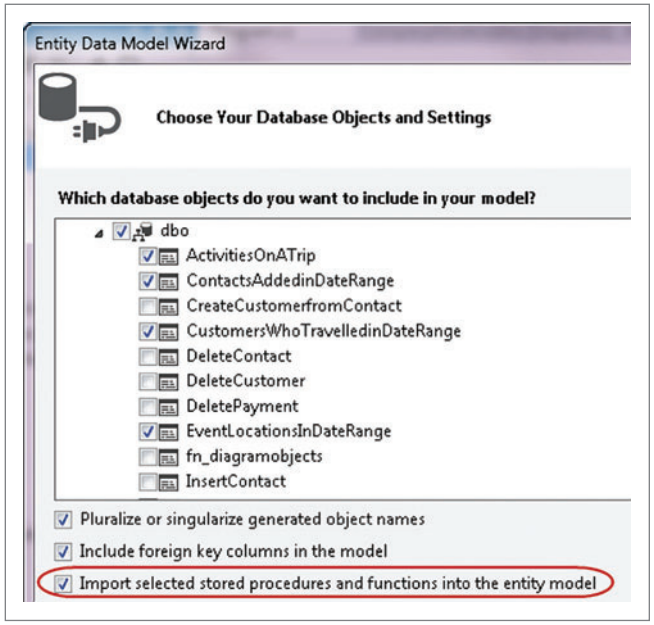


Figure 7 **Creating Function Imports from Stored Procedures**

Center topic, "Reverting Back to ObjectContext Code Generation," at bit.ly/OFjcLa.

Define Enum Types

The Microsoft .NET Framework 4.5 brings enum support to Entity Framework. You can use enums with EDMX and Code First. With EDMX, you need to make the model aware of the enum types so Entity Framework knows how to transform them in order to map them to database types. The Entity Framework designer lets you do this pretty easily.

If you reverse-engineer a database into an EDMX model, the wizard will interpret database columns as .NET types. You'll have to then define your enum types and fix up properties. Here's a quick look at that process.

The Return entity in my model has a property named Reason that maps to an integer. I want to represent Reason as an enum rather than having an extra database table and entity, and I can make this change in the designer. Right-click on the designer background and under Add New, you'll see Enum Type is a new item on the list. Select that to get the Enum Type wizard, where you can define the enum as I've done in **Figure 5**. Enum types aren't visible on the designer surface, but you can always get to them (to view, edit or delete) in the Model Browser.

Now that the model is aware of this enum type, I can use it in my entities. I'll go back to the Return entity and edit the Properties of the Reason property. The type was originally Int32. The new enum type shows up in the Type dropdown list, as shown in **Figure 6**, but you have to scroll down to the bottom to find it. Once you've selected that type, Entity Framework knows how to transform an enum to a database value whether you're writing LINQ queries, retrieving data or updating data.

Here, for example, is a LINQ query that filters on the ReturnReason enum:

```
var returns = context.Returns.Where(
  r => r.Reason == ReturnReason.Broken).ToList();
```

When the query is executed in the database, the Broken enum is transformed to its enum value, 3, as shown in this bit of TSQL:

```
WHERE 3 =  CAST( [Extent1].[Reason] AS int)
```

You may have noticed the Set Flags attribute and Reference external type checkboxes in **Figure 5**. EF supports bit-wise enums, allowing you to combine enums when setting or querying data. Set Flags specifies that the enum will be bit-wise. Pawel Kadluczka from the EF team has a great blog post, "Using Existing Enum Types in Entity Framework 5" (bit.ly/QIUz6y), which provides a detailed explanation of the functionality of the Reference external type checkbox.

Be aware that, as of the time of this writing, WCF Data Services 5 *does not* recognize enum types. Read more about this problem on the WCF Data Services team blog at blogs.msdn.com/astoriateam.

Batch-Import Stored Procedures

Figure 7 shows another feature I remember asking about a long time ago, and I'm quite happy to see it in the Entity Data Model Wizard in Visual Studio 2012. The designer in Visual Studio 2010 lets you map stored procedures to complex types, but you have to do it one stored procedure at a time. That feature still exists, but now there's a shortcut. When you're creating a Database First model,

you can have the wizard create the functions from the stored procedures all at once and the wizard will create the necessary complex types to which the functions will map.

Figure 8 shows the new function imports and their related complex types in the model browser. I typically don't like to use the default names the wizard creates. But it's still faster to let it create these assets for me and then edit the names, than it is to create them myself one by one.

If you want to create function imports from some of the stored procedures but create entity mappings (that is, use InsertContact any time you add a new contact and call SaveChanges) from some others, I recommend you do this in two passes. In the first pass you'd select only the stored procedures that should be imported as functions and their relevant complex types. Then, using the Update Model from Database feature, you can select the procedures you want to use for mapping and ensure that the "Import selected stored procedures …" checkbox isn't checked on this second pass.

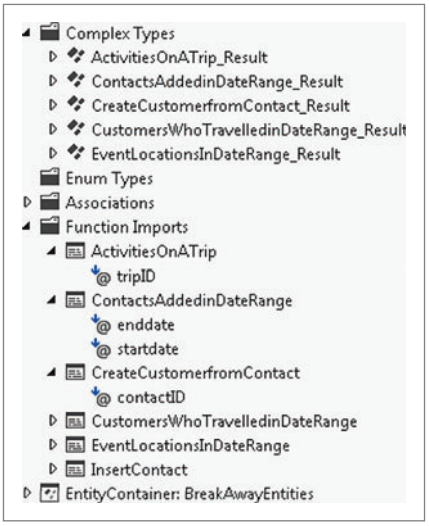


Figure 8 **The New Function Imports**

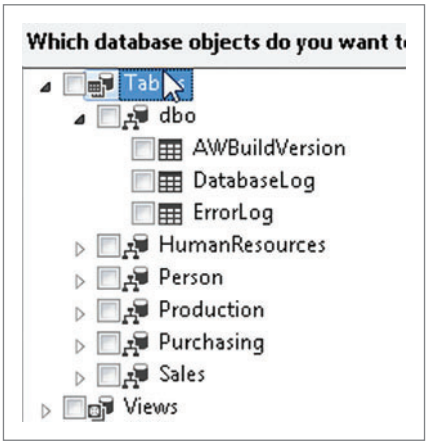


Figure 9 **Grouping Database Objects by Schema Name**

TVF and Geography Support

EF5 supports mapping to table-valued functions (TVFs) and spatial data (such as SqlGeography and SqlGeometry), and the designer also supports these features. You can find out how to use the designer to leverage these mappings by reading the walk-throughs related to TVFs and spatial types at, respectively, bit.ly/QCppJJ and bit.ly/VdbEUP.

As with the enum support, note that WCF Data Services 5 *does not* provide support for the geography types that you can define in your EF5 model.

It's the Little Things

There are a slew of smaller improvements to the Entity Framework Designer that add up to a big overall effect, making use of the designer a smoother experience.

Move Properties In the designer you can reorganize properties in an entity so they're listed in the order you prefer. In earlier versions, the only way to accomplish this was by editing the XML directly.

Schema in Table Selection When you create a Database First model or update a model from the database, the tables, views and other database objects are now organized by schema name. If you've gone to the trouble of organizing your database objects by schema, you'll benefit here by being able to locate objects more easily. And you can easily eliminate objects that aren't part of your domain, such as the dbo schema tables in **Figure 9**.

Column Facet Changes Reflected in Model Update The Update Model from Database Wizard now picks up changes to the column facets—Scale, Precision, MaxLength, Nullable, Unicode and FixedLength—and applies them to the relevant properties in your model. This will be a welcome change for many developers. You'll still have to fix broken mappings when you make changes to column types, though. MSDN has a full description of the effects in the document "Changes Made to an .edmx File by the Update Model Wizard" (bit.ly/PTOwKB). At the time of this writing, the section describing the Column changes does not yet include the new details about the facets.

Highlight Related Entities on Selection This is another little charmer that can help you visually comprehend your model: When you select an entity or association in the designer, the related entities are highlighted as well.

A Note About the Refactor Item on the Context Menu Prerelease versions of Visual Studio 2012 included a designer feature that would refactor dependent code if you renamed an entity or a property. The feature didn't make it into the final release of Visual Studio 2012, but there are some artifacts in the designer that might be a little confusing. The context menu has a Refactor item with Rename in a sub-menu. If you choose Rename from the Refactor menu, a window will pop up indicating that no code will be changed. Because the refactoring won't happen, that window will never list anything.

Robust Improvements

While this article has focused on new features specific to the Entity Framework Designer in Visual Studio 2012, there are other enhancements in EF5 you shouldn't miss. Most notable is how the underlying performance has been improved, thanks to the new Auto-Compiled Queries. You can read about this on the team blog post, "Sneak Preview: Entity Framework 5.0 Performance Improvements" (bit.ly/PLWu5I). Whether you're building your models using the Entity Framework Designer or leveraging Code First to design your models, there's plenty to be excited about with EF5. ■

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.*

Thanks to the following technical expert for reviewing this article: *Lawrence Jones*

5 YEARS OF EXCELLENCE

XCEED
DataGrid
for WPF

Mature, feature-packed, and lightning-fast. The most adopted and trusted WPF datagrid.

IBM®
U2 SystemBuilder™

"IBM U2 researched existing third-party solutions and identified Xceed DataGrid for WPF as the most suitable tool. The datagrid's performance and solid foundation take true advantage of WPF and provide the extensibility required for IBM U2 customers to take their applications to the next level in presentation design and styling."

Vincent Smith
U2 Tools Product Manager at IBM

Microsoft®
Visual Studio® Team System 2010

"Using Xceed DataGrid for WPF in Microsoft Visual Studio System 2010 helped us greatly reduce the time and resources necessary for developing all the data presentation feature we needed. Working with Xceed has been a pleasure."

Norman Guadagno
Director of Product Marketing
for Microsoft Visual Studio Team System

XCEED Professional Themes for WPF

Theme your entire app in minutes. Flawless styles for all official WPF controls.

XCEED DataGrid for Silverlight

Incredible streaming technology. Speed up your app and say goodbye to paging.

XCEED Ultimate ListBox for WPF

NEW

The world's first streaming listbox. Simple, drop-in upgrade to the WPF listbox.

XCEED Ultimate ListBox for Silverlight

NEW

Fast and fluid, with ground-breaking streaming technology.

Visual Studio PARTNER

XCEED MULTI-TALENTED COMPONENTS

BRUNO TERKALY AND
RICARDO VILLALOBOS

Windows Azure Mobile Services: A Robust Back End for Your Device Applications

During the last few years, the developer's landscape has been characterized by three main features: multiple devices and OSes to support; applications that rely on asynchronous Web services, typically hosted in the cloud; and variable traffic that complicates resource planning and provisioning. The new Microsoft Windows Azure Mobile Services (WAMS) simplifies the implementation of software architectures that address this type of environment, as shown in **Figure 1**.

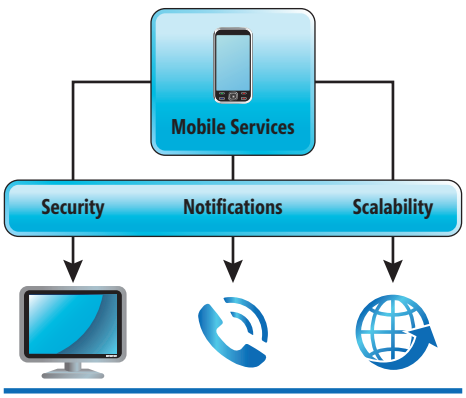


Figure 1 **Typical Architecture to Support Multiple Devices with a Single Web Service**

Even though it's possible to manually create and deploy the required components for this approach to work, what mobile developers really want are endpoints that provide the needed functionality without having to worry about the underlying infrastructure. Having to spend cycles on maintaining structured or semi-structured storage, user authentication or push notifications is a distraction from developing mobile applications.

Keep in mind that we're using the term "mobile application" very loosely here. This client-side (mobile application) technology can target almost anything: a Windows 8 phone or tablet; an iOS iPhone or iPad; or an Android phone or tablet. This is possible due to the fact that WAMS is based on open standards and regular HTTP requests.

Creating Your First Application Based on WAMS

WAMS is currently in preview mode and getting started is just a few clicks away. Simply sign up for a free trial, at bit.ly/azuretestdrive and log in to access the Windows Azure Portal, then click on CREATE A NEW APP.

As you'll see, a window will pop up that allows you to quickly accomplish several things. Here you'll define a URL that will be used by your mobile application to interact with the available services and you'll create a new relational database (though you could choose to use an existing Windows Azure SQL Database). At the same time, a series of REST-based endpoints and services will be created and exposed for mobile clients.

Next, you'll provide a database name, server, login name, password and region, as shown in **Figure 2**.

In just a few moments, you created a new mobile service in one of the reliable, secured, and globally distributed Microsoft datacenters, as shown in **Figure 3**.

Now click on the service name ("brunoterkaly" in **Figure 3**) to access the underlying management dashboard.

Using the Dashboard to Create a Windows 8 Application

Once you access the dashboard, you'll be able to generate a simple Windows 8 application that demonstrates how all the pieces fit together. Notice the link in **Figure 4** that reads "Create a new Windows 8 application."

Click on that link to start the WAMS wizard. To create a basic solution for Windows 8 devices, you'll have to accomplish three essential steps, as follows:

1. **Get the tools.** If you haven't yet installed Visual Studio Express 2012 for Windows 8, do so by clicking on the corresponding link. Also, install the Windows Azure Mobile Services SDK by downloading the bits from bit.ly/TpWfxy. Keep in mind that this last component requires Visual Studio 2012 running on a Windows 8 machine.

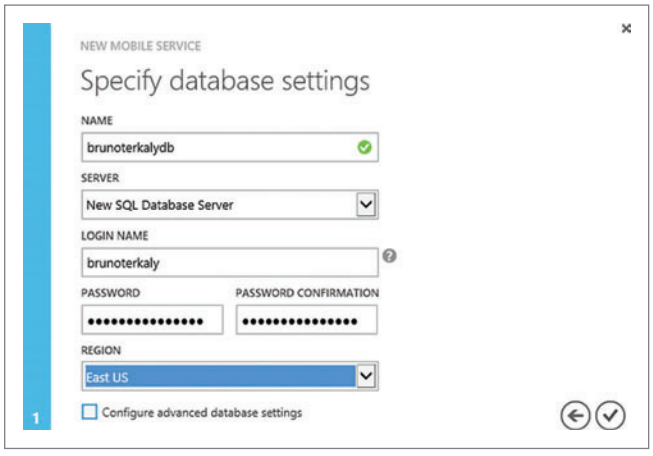


Figure 2 **Specifying Database Settings**

2. **Create a relational table to store your data.** When you click on the Create TodoItem Table button, the wizard will automatically create a table based on the Windows Azure SQL Database you created (or re-used) previously.

   In case you're not familiar with Windows Azure SQL Database, this is a highly scalable service that provides SQL Server capabilities in the cloud. You won't need to do it for this project, but it's actually possible to connect to your newly created database using tools such as SQL Server Management Studio (version 2008 R2 is required), or even from different programming languages.

   The portal even provides a simple interface that allows you to manage your data and table structures. Because Windows Azure SQL Database supports the traditional tabular data stream (TDS) protocol, you can use the exact same tooling as you would with the typical on-premises SQL Server database.

   All Windows Azure SQL Database servers are automatically protected by an external firewall whose rules you can also manage from the portal. The important thing to remember is that the WAMS wizard takes care of all these steps for you.

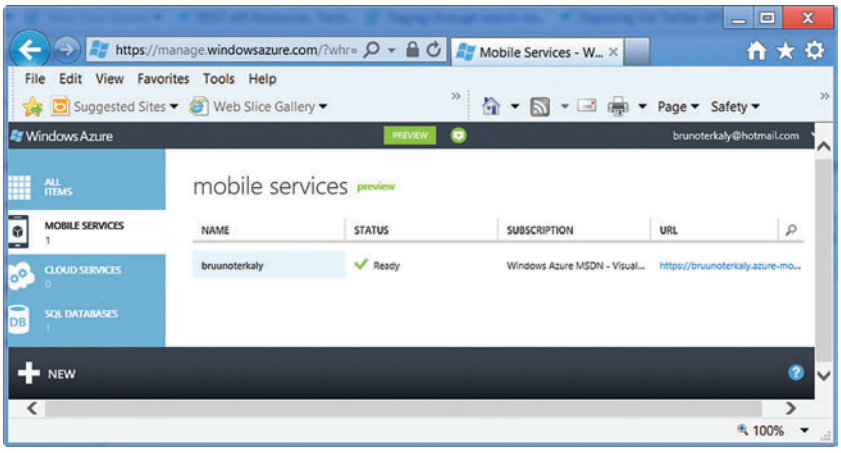


Figure 3 **The Mobile Service Account**

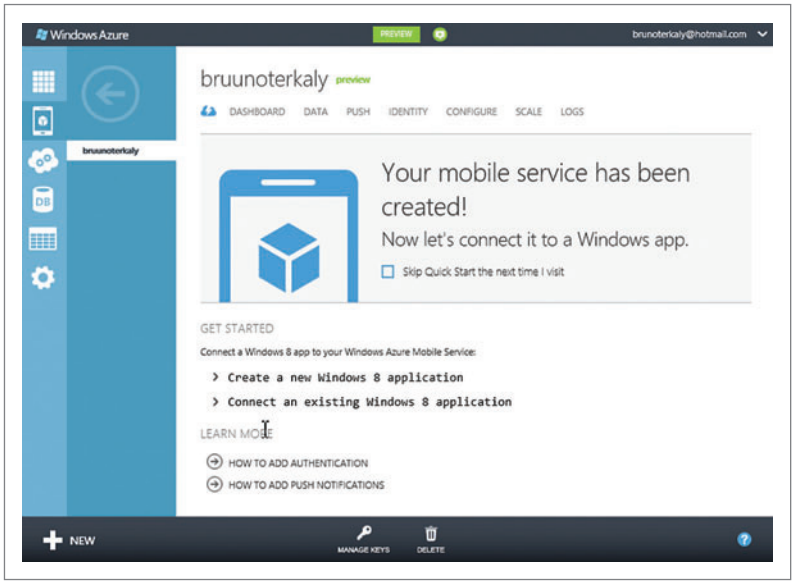


Figure 4 **The Main Portal for Your New Mobile Service Project**

3. **Generate and download the required code for your application.** Now that the required data and endpoints for this project are in place, it's time to look at some code. The final step at the portal allows you to generate a Visual Studio project that automatically takes advantage of the WAMS platform. Here's how to accomplish this:
   1. Choose your programming language, either C# or JavaScript.
   2. When you click Download, you'll be presented with a dialog box that will ask if you want to open or save the generated code. Save the zip file.
   3. After unzipping the code, open the project in Visual Studio 2012.
   4. Now run the code to see how it works. From the DEBUG menu, choose START DEBUGGING. Enter a TodoItem and hit Save, then Refresh, as shown in **Figure 6**.
   5. You can open the generated database using a tool such as SQL Server Management Studio (or the management tool in the portal) to review the data. You'll notice that a new todo item has been added to the corresponding table.

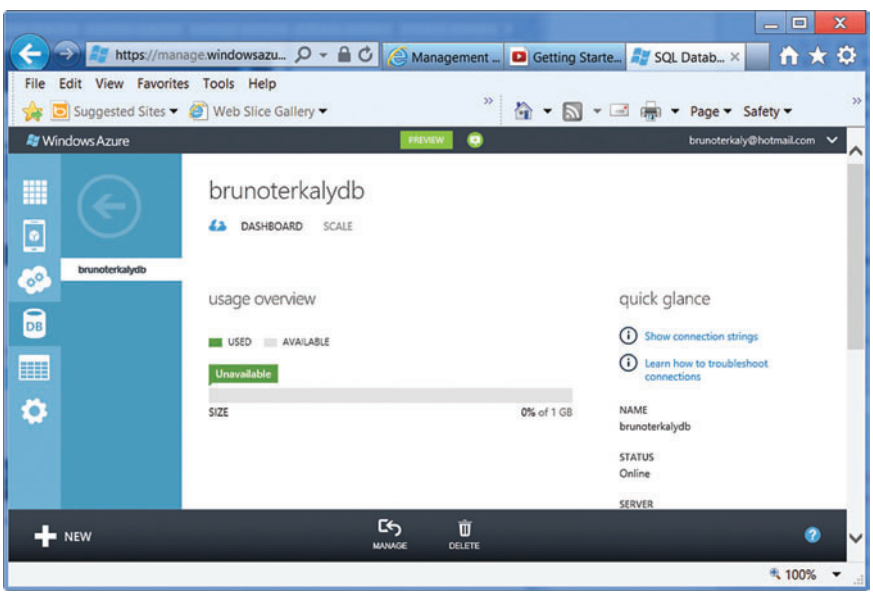


Figure 5 **The Database Dashboard**

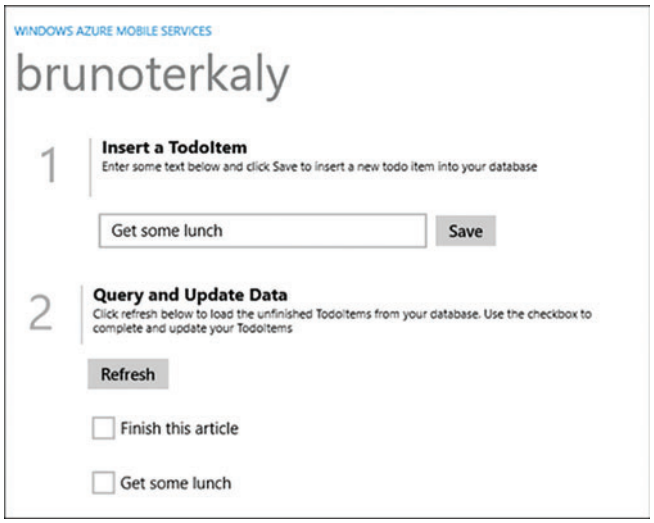


Figure 6 **Running the Application**

Exploring the Code

Let's take a look under the covers to see exactly how this works. The main location for all of this code is in MainPage.xaml.cs. Go to Solution Explorer and double-click on MainPage.xaml.cs to see the code shown in **Figure 7**.

Here's what's happening in the code:

- Lines 19 to 28 create the TodoItem, an object that represents the data we want to save to Windows Azure SQL Database. It represents the same data structure as the underlying table in the database, but they don't need to be the same. The DataMember attributes affect how the item is serialized as well as the name of the actual columns in the Windows Azure SQL Database. This allows you to have a column name in the database that's different than the Property name in the Microsoft .NET Framework.

The Windows Azure Mobile Service can be accessed from practically any environment, because HTTP requests and RESTful calls are widely supported by multiple platforms and programming languages.

- Line 35 creates a collection of items that get displayed in a list control in the main page of the application. The collection is bound to an underlying list control. It's populated from the object on line 37.
- Line 37 creates an IMobileServiceTable of TodoItems named todoTable. The todoTable object acts as a wrapper around the RESTful HTTP API exposed by WAMS and allows you to use client-side LINQ queries rather than manual REST API HTTP calls to query the data in the Windows Azure SQL Database TodoItem table. The result of the LINQ queries are saved into the items collection created earlier, and later bound to the ListView for display. The same todoTable object also allows asynchronous insert, update and delete operations on the Windows Azure SQL Database TodoItem table. This is all code that usually you'd have to write yourself. In short, the developer is saved from having to write a lot of plumbing code to perform typical create, read, update and delete operations.
- Line 51 asynchronously inserts todo items into the table and updates the items collection.
- Line 61 updates the items collection with the latest data from the todo table and binds that data to the list control.
- Line 69 issues asynchronous updates to the underlying SQL database table and removes that item from the items collection.
- Line 74 refreshes the list control with data from the database.
- Line 80 inserts a new entry into the todo table.
- Line 87 updates the underlying table to reflect the fact that a check box has been checked.

Figure 7 **The TodoItem Class**

```
19  public class TodoItem
20  {
21    public int Id { get; set; }
22
23    [DataMember(Name = "text")]
24    public string Text { get; set; }
25
26    [DataMember(Name = "complete")]
27    public bool Complete { get; set; }
28  }
29
30  public sealed partial class MainPage : Page
31  {
32    // MobileServiceCollectionView implements ICollectionView
33    // (useful for databinding to lists) and is integrated with your
34    // MobileService to make it easy to bind your data to the ListView
35    private MobileServiceCollectionView<TodoItem> items;
36
37    private IMobileServiceTable<TodoItem> todoTable =
38      App.MobileService.GetTable<TodoItem>();
42
43    public MainPage()...
50
51    private async void InsertTodoItem(TodoItem todoItem)...
60
61    private void RefreshTodoItems()…
68
69    private async void UpdateCheckedTodoItem(TodoItem item)...
73
74    private void ButtonRefresh_Click(object sender, RoutedEventArgs e)...
79
80    private void ButtonSave_Click(object sender, RoutedEventArgs e)...
86
87    private void CheckBoxComplete_Checked(object sender, RoutedEventArgs e)...
88
89    protected override void OnNavigatedTo(NavigationEventArgs e)...
91  }
```

The App.xaml.cs file is also worth exploring, because it stores the startup code. The client object of type MobileServiceClient abstracts away the complexities of making a RESTful service call, which requires an application key (generated when the Windows Azure Mobile Service is created). You can get the application

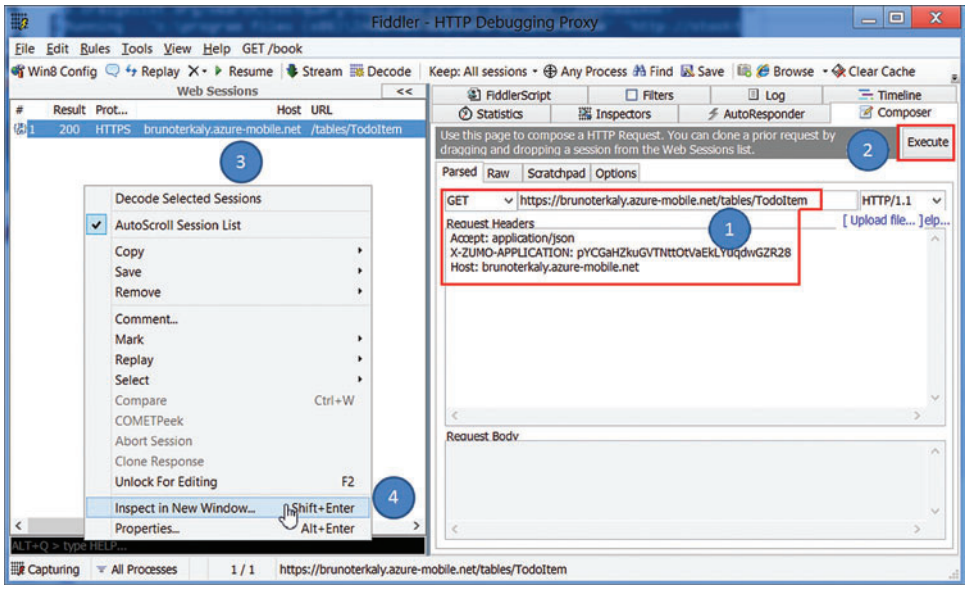


Figure 8 **Analyzing the Generated HTTP Requests Using Fiddler**

key either at the Windows Azure Mobile Services Portal or in the App.xaml.cs file:

```
sealed partial class App : Application
  {
    // This MobileServiceClient has been configured to
    // communicate with your Mobile Service URL and
    // application key; you're all set to start working with
    // your Mobile Service!
    public static MobileServiceClient MobileService =
      new MobileServiceClient(
      "https://brunoterkaly.azure-mobile.net/",
      "rsECynSsGJmPiHmydyTGEYOSlrbCgr27"
);
```

How Requests Get Executed

This is where things get interesting. It's possible to analyze the HTTP requests generated by the Windows 8 client using a Web debugging proxy tool such as Fiddler (which can be downloaded for free at fiddler2.com). Before using Fiddler for this purpose, note that you need to allow Fiddler to capture and decrypt HTTP traffic. In Fiddler, go to Tools | Fiddler Options and then switch to the HTTPS tab and turn on the checkboxes for Capture HTTPS CONNECTs and Decrypt HTTPS traffic.

After starting the tool, navigate to the Composer tab, and follow these steps, as shown in **Figure 8** and **Figure 9**:

1. Select the verb, such as GET, and then enter the URL (for our project it's https://brunoterkaly.azure-mobile.net/tables/TodoItem), and the corresponding header; for example, https://brunoterkaly.azure-mobile.net/tables/TodoItem.
2. Hit Execute.
3. Right-click on the session in the left pane.
4. Choose Inspect in New Window.
5. The Web session pane will appear. Select the Response tab.

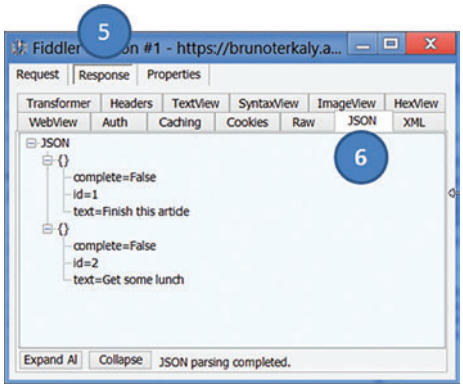


Figure 9 **The JSON Code Returned from WAMS**

6. Select the JSON tab. Now you'll be able to see the data coming from the Windows Azure Mobile Service you created previously. This is the same data you saw in the SQL Server database.

This essentially means that the Windows Azure Mobile Service can be accessed from practically any environment, because HTTP requests and RESTful calls are widely supported by multiple platforms and programming languages. Note that the Windows Azure Mobile Services SDK for Windows 8 makes this easier to use in Windows 8 apps. Also, client libraries will soon be available for iOS and Android.

Wrapping Up

Windows Azure Mobile Services simplifies the life of mobile developers by automating the steps required to create a back end based on open standards, supporting multiple types of devices and OSes. Moreover, WAMS provides a reliable and secure infrastructure in the cloud that can be scaled based on traffic and demand, allowing programmers to concentrate on optimizing their apps and data.

There's more to learn, of course, especially in the areas of push notification and authentication. Push notification makes it possible to easily send notifications to your Windows 8 app without writing, testing or managing back-end infrastructure code. In addition, WAMS eliminates the need to write, configure, and test custom authentication and user management solutions for your Windows 8 apps. We look forward to addressing these technologies in future columns. ■

Bruno Terkaly *is a developer evangelist for Microsoft. His depth of knowledge comes from years of experience in the field, writing code using a multitude of platforms, languages, frameworks, SDKs, libraries and APIs. He spends time writing code, blogging and giving live presentations on building cloud-based apps, specifically using the Windows Azure platform and just wrote a five-part series on iOS and Windows Azure Mobile Services, available at bit.ly/UPXGdV.*

Ricardo Villalobos *is a seasoned software architect with more than 15 years of experience designing and creating applications for companies in the supply chain management industry. Holding different technical certifications, as well as a master's degree in business administration from the University of Dallas, he works as a cloud architect in the Windows Azure CSV incubation group for Microsoft.*

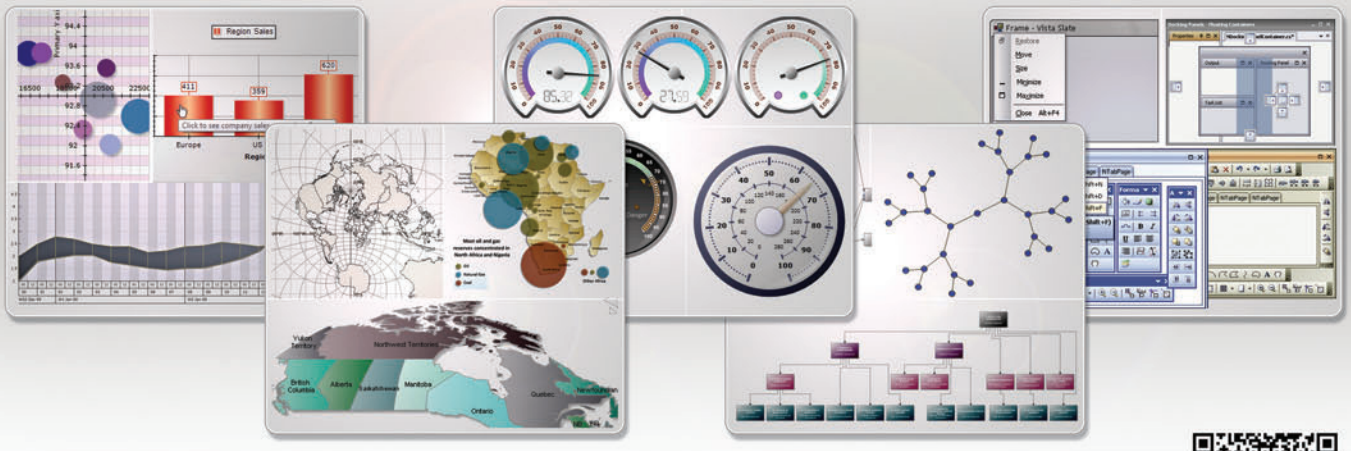
Thanks to the following technical expert for reviewing this article: *Bret Stateham*

NEVRON
visualize your success

NEVRON
GAUGE for .NET, SSRS, SharePoint

NEVRON
CHART for .NET, SSRS, SharePoint

NEVRON
USER INTERFACE for .NET



NEVRON
MAP for .NET

NEVRON
DIAGRAM for .NET

2012vol.1 is here

Features new ThinWeb controls with JQuery and ASP.NET MVC integration, SQL Server and SharePoint "Expressions Everywhere" and more.

Nevron components integrate seamlessly in **Web** and **Desktop .NET** applications, **SQL Server Reporting Services 2005/2008** reports and **SharePoint 2007/2010** portals and deliver an unmatched set of enterprise-grade features. That is why Nevron is the trusted vendor by many Fortune 500 companies for their most demanding data visualization needs.

Download your free evaluation from

www.nevron.com

DEVELOPERS

.NET

IT PROFFESIONALS

SharePoint

SSRS

Microsoft .NET ASP.net AJAX Microsoft SharePoint 2007/2010 Microsoft SQL Server 2005/2008 Microsoft Visual Studio PARTNER

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

YOUR MAP TO THE .NET DEVELOPMENT PLATFORM

BIRDS OF A FEATHER CODE TOGETHER

Orlando, FL | **December 10-14**

Royal Pacific Resort at Universal Orlando | **vslive.com/orlando**

Don't miss your chance for great .NET education in 2012 at Visual Studio Live! Orlando.

- **Mobile**
- **Visual Studio / .NET**
- **Web / HTML5**
- **Windows 8 / WinRT**
- **WPF / Silverlight**

2011 Sold Out...

Register Today

vslive.com/orlando

Use Promo Code VSLNOV

ROYAL PACIFIC RESORT LIVE! DECEMBER ORLANDO 10 - 14 LIVE!

GOLD SPONSORS

axosoft · New Relic. · Symantec. · VEEAM

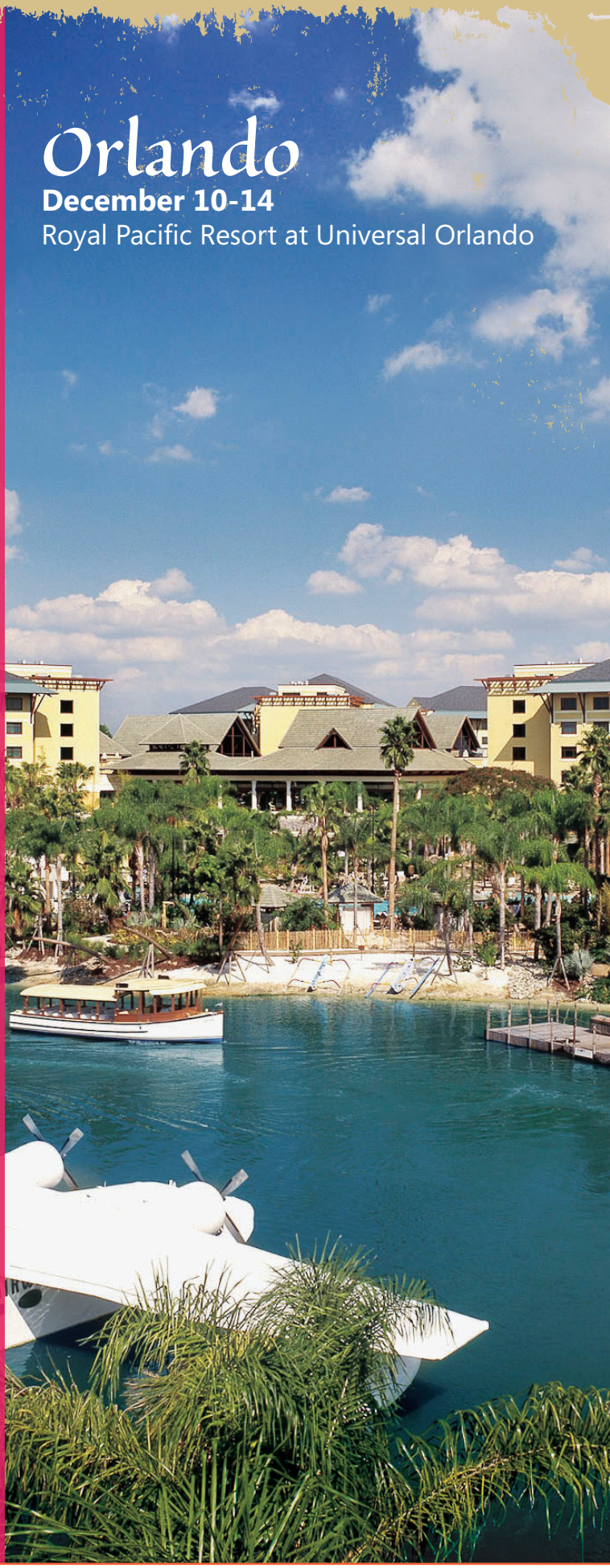
SUPPORTED BY

Microsoft · msdn magazine · Visual Studio MAGAZINE · Redmond MAGAZINE · VIRTUALIZATION REVIEW

Orlando

December 10-14

Royal Pacific Resort at Universal Orlando



Mobile Track Sessions Include:

- What's New in Windows Phone 8 for Developers
- Reach the Mobile Masses with ASP.NET MVC 4 and jQuery Mobile
- Development Survival Guide for the .NET Guy

Visual Studio / .NET Track Sessions Include:

- Workshop: Services - Using WCF and ASP.NET Web API
- Workshop: TFS/ALM
- What's New in Visual Studio 2012
- What's New in the .NET 4.5 BCL
- Whats New in Azure for Developers

Web / HTML5 Track Sessions Include:

- Workshop: Mastering ASP.NET MVC4 in Just One Day
- Building Single Page Web Applications with HTML5, ASP.NET MVC4, Upshot.js and Web API
- HTML5 for Business: Forms and Input Validation
- JavaScript and jQuery for .NET Developers
- Making HTML5 Work Everywhere

Windows 8 / WinRT Track Sessions Include:

- Workshop: Build a Windows 8 Application in a Day
- Build Windows 8 Apps using Windows Online Services
- WinRT Services
- Smackdown: Metro Style Apps vs. Websites
- Using Azure with Windows Phone and Windows 8!

WPF / Silverlight Track Sessions Include:

- Workshop: XAML User Experience Design
- Leveraging XAML for a Great User Experience
- Implementing the MVVM Pattern in WPF
- Building High Performance, Human-Centered Interactive Data Visualizations Dashboards
- Navigation and UI Shells for XAML Applications

MEDIA SPONSOR PRODUCED BY

THE CODE PROJECT WWW.CODEPROJECT.COM

1105 MEDIA

Your 2012 Visual Studio Live! Orlando Speakers

Todd Anglin
Chief Evangelist, Telerik

Sam Bisbee
Director of Technical Business Development, Cloudant

Jason Bock
MVP (C#), Principal Consultant, Magenic

Keith Burnell
Senior Software Engineer, Skyline Technologies

Miguel Castro
Architect, IDesign

Tiberiu Covaci
Independent Trainer

Rob Daigneau
Principal, ArcSage LLC

Benjamin Day
Architect / Trainer / Coach, Benjamin Day Consulting, Inc.

Marcel de Vries
Technology Manager, Info Support

Ben Dewey
Senior Software Consultant, Tallan, Inc

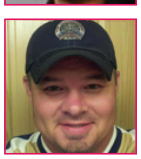
Ben Hoelting
Software Architect, Aspenware

Billy Hollis
Next Version Systems

Philip Japikse
Patterns & Practices Evangelist, Telerik

Nick Landry
Senior Product Manager, Infragistics

Vishwas Lele
Architect, AIS



Greg Levenhagen
Senior Software Engineer, Skyline Technologies

Rockford Lhotka
CTO, Magenic

Steve Michelotti
Architect/Developer, Applied Information Sciences

Ted Neward
Architectural Consultant, Neudesic

John Papa
JohnPapa.net, LLC

Brian Randell
Senior Consultant, MCW Technologies

Elad Shaham
Senior Consultant and Instructor, Sela Group

Connect with Visual Studio Live!

http://twitter.com/vslive – @VSLive

http://www.facebook.com – Search "vslive"

http://www.linkedin.com – Join the "vslive" group!

Register at vslive.com/orlando

Use Promo Code VSLNOV

VISUAL STUDIO LIVE! ORLANDO AGENDA AT-A-GLANCE

Web / HTML5	Mobile	WPF / Silverlight	Windows 8 / WinRT	Visual Studio / .NET

START TIME	END TIME	Visual Studio Live! Pre-Conference Workshops: Monday, December 10, 2012 *(Separate entry fee required)*			
6:30 am	8:00 am	Pre-Conference Workshop Registration – Coffee and Morning Pastries			
8:00 am	12:00 pm	VSM1 **Workshop: XAML User Experience Design** *Billy Hollis*	VSM2 **Workshop: Services - Using WCF and ASP.NET Web API** *Miguel Castro*	VSM3 **Workshop: Build a Windows 8 Application in a Day** *Rockford Lhotka*	
12:00 pm	1:00 pm	Lunch			
1:00 pm	5:00 pm	VSM1 **Workshop: XAML User Experience Design** *Billy Hollis*	VSM2 **Workshop: Services - Using WCF and ASP.NET Web API** *Miguel Castro*	VSM3 **Workshop: Build a Windows 8 Application in a Day** *Rockford Lhotka*	
5:00 pm	7:00 pm	EXPO Preview			
7:00 pm	8:00 pm	Live! 360 Keynote			

START TIME	END TIME	Visual Studio Live! Day 1: Tuesday, December 11, 2012			
7:00 am	8:00 am	Registration – Coffee and Morning Pastries			
8:00 am	9:00 am	Visual Studio Live! Keynote			
9:15 am	10:30 am	Live! 360 Keynote			
10:30 am	11:00 am	Networking Break • Visit the EXPO			
11:00 am	12:15 pm	VST1 **What's New in Azure for Devs** *Vishwas Lele*	VST2 **Leveraging XAML for a Great User Experience** *Billy Hollis*	VST3 **Intro to Metro** *Miguel Castro*	VST4 **What's New in Visual Studio 2012** *Rob Daigneau*
12:15 pm	2:00 pm	Lunch • Visit the EXPO			
2:00 pm	3:15 pm	VST5 **Azure Web Sites (Antares)** *Steve Michelotti*	VST6 **Implementing the MVVM Pattern in WPF** *Miguel Castro*	VST7 **Smackdown: Metro Style Apps vs. Websites** *Ben Dewey*	VST8 **What's New in Windows Phone 8 for Developers** *Nick Landry*
3:15 pm	4:15 pm	Networking Break • Visit the EXPO			
4:15 pm	5:30 pm	VST9 **In Depth Azure IaaS** *Vishwas Lele*	VST10 **Building High Performance, Human-Centered Interactive Data Visualizations Dashboards** *Nick Landry*	VST11 **Going from Silverlight or WPF to Metro** *Greg Levenhagen*	VST12 **What's New in the .NET 4.5 BCL** *Jason Bock*
5:30 pm	7:30 pm	Exhibitor Reception			

START TIME	END TIME	Visual Studio Live! Day 2: Wednesday, December 12, 2012			
7:00 am	8:00 am	Registration – Coffee and Morning Pastries			
8:00 am	9:00 am	Visual Studio Live! Keynote			
9:15 am	10:30 am	VSW1 **JavaScript and jQuery for .NET Developers** *John Papa*	VSW2 **Navigation and UI Shells for XAML Applications** *Billy Hollis*	VSW3 **Filling Up Your Charm Bracelet** *Ben Dewey*	VSW4 **To Be Announced**
10:30 am	11:00 am	Networking Break • Visit the EXPO			
11:00 am	12:15 pm	VSW5 **Applying JavaScript Patterns** *John Papa*	VSW6 **Static Analysis in .NET** *Jason Bock*	VSW7 **WinRT Services** *Rockford Lhotka*	VSW8 **Reach the Mobile Masses with ASP.NET MVC 4 and jQuery Mobile** *Keith Burnell*
12:15 pm	1:45 pm	Round Table Lunch • Visit the EXPO			
1:45 pm	3:00 pm	VSW9 **HTML5 for Business: Forms and Input Validation** *Todd Anglin*	VSW10 **Controlling ASP.NET MVC 4** *Philip Japikse*	VSW11 **Using Azure with Windows Phone and Windows 8!** *Greg Levenhagen*	VSW12 **iOS Development Survival Guide for the .NET Guy** *Nick Landry*
3:00 pm	4:00 pm	Networking Break • Visit the EXPO • Expo Raffle			
4:00 pm	5:15 pm	VSW13 **Making HTML5 Work Everywhere** *Todd Anglin*	VSW14 **I'm Not Dead Yet! AKA the Resurgence of WebForms** *Philip Japikse*	VSW15 **From a New Win8 Project to the Store** *Elad Shaham*	VSW16 **Creating Restful Web Services with the Web API** *Rob Daigneau*

START TIME	END TIME	Visual Studio Live! Day 3: Thursday, December 13, 2012			
7:00 am	8:00 am	Registration – Coffee and Morning Pastries			
8:00 am	9:15 am	VSTH1 **Identify & Fix Performance Problems with Visual Studio Ultimate** *Benjamin Day*	VSTH2 **Patterns for Parallel Programming** *Tiberiu Covaci*	VSTH3 **Windows 8 Metro Style Apps for the Enterprise** *Ben Hoelting*	VSTH4 **How to Take WCF Data Services to the Next Level** *Rob Daigneau*
9:30 am	10:45 am	VSTH5 **Building Single Page Web Applications with HTML5, ASP.NET MVC4, Upshot.js and Web API** *Marcel de Vries*	VSTH6 **ALM Features in VS12** *Brian Randell*	VSTH7 **Build Windows 8 Apps using Windows Online Services** *Rockford Lhotka*	VSTH8 **To Be Announced**
11:00 am	12:15 pm	VSTH9 **Busy .NET Developer's Guide to Node.js** *Ted Neward*	VSTH10 **Intellitrace, What is it and How Can I Use it to My Benefit?** *Marcel de Vries*	VSTH11 **Expression Blend 5 For Developers : Design Your XAML or HTML5\CSS3 UI Faster** *Ben Hoelting*	VSTH12 **To Be Announced**
12:15 pm	1:30 pm	Lunch			
1:30 pm	2:45 pm	VSTH13 **A Practical Look at the NOSQL and Big Data Hullabaloo** *Sam Bisbee*	VSTH14 **Team Foundation Server 2012 Builds: Understand, Configure, and Customize** *Benjamin Day*	VSTH15 **Making your Windows 8 App Come to Life, Even When It's Not Running** *Elad Shaham*	VSTH16 **EF Code First Magic Unicorn Edition and Beyond** *Keith Burnell*
3:00 pm	4:15 pm	VSTH17 **Introduction to OAuth** *Ted Neward*	VSTH18 **Azure Hosted TFS** *Brian Randell*	VSTH19 **No User Controls Please: Customizing Metro Style Controls using XAML** *Elad Shaham*	VSTH20 **The LINQ Programming Model** *Marcel de Vries*
4:30 pm	5:45 pm	Live! 360 Conference Wrap-up			

START TIME	END TIME	Visual Studio Live! Post-Conference Workshops: Friday, December 14, 2012 *(Separate entry fee required)*	
7:00 am	8:00 am	Post-Conference Workshop Registration – Coffee and Morning Pastries	
8:00 am	12:00 pm	VSF1 **Workshop: Mastering ASP.NET MVC4 in Just One Day** *Tiberiu Covaci*	VSF2 **Workshop: ALM with Visual Studio 2012 and Team Foundation Server 2012** *Brian Randell*
12:00 pm	1:00 pm	Lunch	
1:00 pm	5:00 pm	VSF1 **Workshop: Mastering ASP.NET MVC4 in Just One Day** *Tiberiu Covaci*	VSF2 **Workshop: ALM with Visual Studio 2012 and Team Foundation Server 2012** *Brian Randell*

For the complete session schedule and full session descriptions, please check the Visual Studio Live! Orlando web site at ***vslive.com/orlando***

**Speakers and Sessions Subject to Change*

Web to Windows 8: Security

Tim Kulp

Years ago I thought it would be a good idea to learn how to play golf. Before I signed up for lessons at my local driving range, I had never picked up a golf club. At my first lesson, the instructor asked me if I had ever had lessons before or ever tried to play golf. When I told him no, he said, "Good! We won't have any old habits to get out of your swing."

Web developers transitioning from the browser to Windows Store apps bring certain habits with them. While Web developers can tap in to their existing knowledge of JavaScript, some capabilities are new and require a shift in thinking. Security is one such fundamental difference. Many Web developers are in the habit of handing applications security off to the server because of reasons such as, "Why bother? JavaScript can be easily bypassed." On the Web client side, security features are seen as improving usability without adding value to the overall security of the Web application.

This article discusses:

- Web developers transitioning to Windows 8
- Building security into Windows Store apps
- Validating input
- Storing sensitive data
- Using local and Web contexts
- Communicating between contexts

Technologies discussed:

Windows 8, JavaScript, HTML5

With Windows 8, JavaScript plays an important part in the overall security of your app by providing the tools necessary to secure data, validate input and separate potentially malicious content. In this article, I show you how you can adjust some of the habits you bring from Web development so that you can produce more secure Windows Store apps using HTML5, JavaScript and the security features of the Windows Runtime.

Input Validation

Web developer says: JavaScript validation is for usability and doesn't add to the application's security.

Windows 8 developer says: Validation with HTML5 and JavaScript is your first line of defense against malicious content getting in to your app.

For traditional Web applications, JavaScript is often just a gateway to the server. All important actions with the data, such as input validation and storage, occur on the server. Malicious attackers can disable JavaScript on their browser or directly by submitting handcrafted HTTP requests to circumvent any client-side protections. In a Windows Store app, the developer can't rely on a server to clean user input prior to acting on the data because there's no server. When it comes to input validation, JavaScript and HTML5 are on their own.

In software security, input validation is a critical component for data integrity. Without it, attackers can use every input field as a possible attack vector into the Windows Store app. In the second edition of "Writing Secure Code" (Microsoft Press, 2003), authors Michael

Howard and Steve Lipner provide what has become a mantra for managing input: "All input is evil until proven otherwise."

You shouldn't trust data until it's proven to conform to "known good" data. When building an app, the developer knows what data from a specific field should look like (that is, an allow list) or at the very least what it shouldn't have (that is, a deny list). In the world of input validation, always use an allow list when possible to restrict input to known good data. By allowing only data that you know is good, you reduce the possibility of missing a new or unknown way to represent bad data.

Constrain, Reject and Sanitize

How do developers reduce risk to their users by limiting input to known good data? They use the three stages of input validation shown in **Figure 1** to reduce the risk of malicious content getting in to their app.

Input validation begins with constraining data to what is "known good." Web developers familiar with HTML5 can use their existing knowledge of its new input types and attributes to constrain data coming into their Windows Store apps. The key difference between the Web and Windows 8 is that a Windows Store app doesn't have a server behind the scenes that checks input. Constraining data must happen in HTML5 or JavaScript.

Using HTML5, each field can easily be constrained to known good data. To illustrate examples in this article, I use the fictitious Contoso Health app, which stores personal health information for users. The Profile page of this app captures the user's name, e-mail address, weight and height and provides a notes field for general information. As the developer, I know (in general) what good data looks like for each of these fields:

- **Name:** Alphabetic characters with a few special characters not to exceed 45 total characters. The name criteria is based on the target market for the app, the U.S. market.
- **E-mail Address:** Input must be a valid e-mail address format.
- **Weight and Height:** Numbers with associated labels to show data is in feet and inches and in pounds.
- **Notes:** HTML content using the standard Contoso HTML editor.

For the Name input element, I need to limit what characters are valid for the field as well as how long the value can be. I can do this using two new attributes of the input tag: pattern and title.

Pattern is a regular expression to which the data entered must conform. MSHTML (the rendering engine used for HTML5 apps in Windows 8) verifies that data entered into the field matches the regular expression. If the user enters data that doesn't conform to the regular expression pattern, submitting the form will fail and the user will be directed to correct the invalid field. For example, the Name field can be composed of alpha characters and spaces, and it must be three to 45 characters long. The following pattern value supports this:

```
<input type="text" id="txtName" name="txtName"
  pattern="^[A-Za-z ]{3,45}$" title="" />
```

Title is used to inform the user of what the system is expecting. In this case, something such as "Name must be three to 45 characters long using alphabetic characters or spaces" would explain the expected pattern. Nothing is more frustrating to users than having invalid input without knowing what valid input is. Be nice to your users and let them know what's allowed. The title attribute does just that; it's the explanation message that shows what's expected in the field.

Patterns for the data fields including acceptable characters and length can be difficult to determine. You can find sample regular expressions in many great online resources, but always consult with your organization's security team to see whether there is a standard to which you must conform. If you don't have a security team or if your security team doesn't have standards, resources such as RegExLib.com provide an excellent library of regular expressions you can use for your data validations.

Some fields are specific data types, such as numbers, dates and e-mail addresses. HTML5 comes to the rescue again with an army of new input types, such as email, phone, date, number and many more. Using these data input types, MSHTML checks whether what the user entered is valid data, without any regular expressions or JavaScript code necessary. The input element's type attribute handles the new data types. (You can find more types and their uses at bit.ly/OH1xFf.) For example, to capture an e-mail address for the Profile page, I would set the type attribute to be email, as in the following example:

```
<input type="email" id="txtEmail" name="txtEmail" />
```

This field accepts a value only if it conforms to the format of a valid e-mail address. If MSHTML doesn't recognize input as a valid e-mail address, a validation error displays on the field when the user attempts to submit the form. Using the new input types of HTML5 constrains the data to what you're expecting without the hassle of complex JavaScript validation.

Some of the new input types also allow range restrictions using the new min and max attributes. As an example, because of the business rules, the people in our app must have a height between 3 and 8 feet. The following range restrictions can be used on the height field:

```
<input type="number" id="txtHeight" name="txtHeight" min="3" max="8" />
```

The examples provided use the four techniques to constrain data with the HTML5 input tag. By validating length (using a pattern), format (again, using the pattern), data type (using the new input types) and range (using min/max), you can constrain the data to be known good data. Not all attributes and types prompt you to correct them prior to submission. Make sure you validate your form's contents with the checkValidity method (bit.ly/SgNgnA) just as you would Page.IsValid in ASP.NET. You might be wondering whether you can constrain data like this just by using JavaScript. Yes, you can, but using the HTML5 attributes reduces the overall

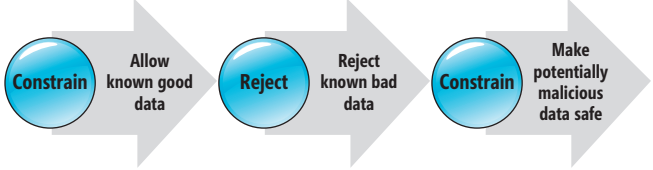


Figure 1 **Input Validation (Image Based on Figure 4.4 from Chapter 4, "Design Guidelines for Secure Web Applications," of "Improving Web Application Security: Threats and Countermeasures" at** bit.ly/emYI5A)

code the developer needs to manage by handing all the heavy lifting over to the MSHTML engine.

Reject denies known bad (that is, a deny list) input. A good example of reject is creating a deny list of IP addresses that can't connect to your Web application. Deny lists are useful when you have a somewhat fixed scope defined for what you want to block. As an example, consider sending e-mail to a group such as your development team and then specifically removing individuals from the development team e-mail list. In this example, you know which e-mail addresses you want to deny from the development team list. For secure software, you want to focus on constrain (an allow list) over reject (a deny list). Always remember that known bad data changes constantly as attackers find ever more creative ways to circumvent software defenses. In the preceding example, imagine new developers joining the development team and needing to vet whether they should be included in the e-mail. Constraints are much easier to manage in the long run and provide a more maintainable list as opposed to the thousands of items in a deny list.

> Don't forget best practices surrounding sensitive data, such as accessing the data only when necessary and keeping sensitive data out of the cache.

Sometimes data contains both known good and known bad data. An example of this is HTML content. Some tags are approved to display while others are not. The process of filtering out or disabling the known bad data and allowing the approved data is known as sanitizing the input. The notes field in the Contoso Health app is a great example of this. Users can enter HTML tags through an HTML editor, but only certain HTML tags are rendered when the input is displayed in the app. Sanitizing input takes data that could be malicious and makes it safe by stripping unsafe content and rendering inert what isn't explicitly approved. Windows Store apps can do this if you set the value of an HTML element using innerText (instead of innerHTML), which renders the HTML content as text instead of interpreting it as HTML. (Note that if the app sets the innerText of a script tag to JavaScript, executable script is produced.) JavaScript also provides another useful tool for sanitization: toStaticHTML.

Here's sample code from the Profile page's btnSave_Click handler:

```
function btnSave_Click(args) {
  var taintedNotes = document.getElementById("txtNotes").value;
  var sanitizedNotes = window.toStaticHTML(taintedNotes);
  document.getElementById("output").innerHTML = sanitizedNotes;
}
```

If the user enters the string

```
<strong>testing!</strong><script>alert("123! ");</script>
```

to txtNotes, the window.toStaticHTML method strips out the script tag and leaves only the approved strong tag. Using toStaticHTML strips any tag that isn't on the approved safe list (another example of using an allow list), as well as any attribute that is unknown. Only known good data is kept in the output of the toStaticHTML method. You can find a complete listing of approved tags, attributes, CSS rules and properties at bit.ly/KNnjpF.

Input validation reduces the risk of malicious content entering the system. Using HTML5 and toStaticHTML, the app can restrict input to known good data and remove or disable possibly malicious content without server intervention.

Now that Contoso Health is getting valid data, what do we do with sensitive data such as medical or financial information?

Sensitive Data Storage

Web developer says: Never store sensitive data on the client because secure storage is unavailable.

Windows 8 developer says: Sensitive data can be encrypted and securely stored through the Windows Runtime.

In the previous section, the Contoso Health app retrieved general profile information. As development continues, a medical history form is requested by the business sponsor. This form captures medical events that occur throughout a user's life, such as the most recent doctor's visit. Old rules for Web development say that storing sensitive information such as a user's medical history on the client is a bad idea because of the possible exposure of the data. In Windows Store app development, sensitive data can be stored locally using the security features of the Windows Runtime.

To protect the user's medical history, Contoso Health uses the WinRT Data Protection API. Encryption shouldn't be the only part of your data-protection strategy (think Defense in Depth: layers of security instead of a single defense, such as using only encryption). Don't forget other best practices surrounding sensitive data, such as accessing the data only when necessary and keeping sensitive data out of the cache. A great resource that lists many considerations for sensitive data is the MSDN Library article, "Improving Web Application Security: Threats and Countermeasures" (bit.ly/NuUe6w). Although this document focuses on Web development best practices, it provides a lot of excellent foundation knowledge that you can apply to any type of development.

The Medical History page in the Contoso Health app has a button named btnAddItem. When the user clicks btnAddItem, the app encrypts data entered into the Medical History form. To encrypt the Medical History information, the app uses the built-in WinRT Data Protection API. This simple encryption system allows developers to encrypt data quickly without the overhead of key management. Begin with an empty event handler for the btnAddItem click event. Then Contoso Health collects the form information and stores it in a JSON object. Inside the event handler, I add the code to quickly build out the JSON object:

```
var healthItem = {
  "prop1": window.toStaticHTML(document.getElementById("txt1").value),
  "prop2": window.toStaticHTML(document.getElementById("txt2").value)
};
```

The healthItem object represents the Medical History record the user has entered into the form. Encrypting healthItem begins with instantiating a DataProtectionProvider:

Does your Team do more than just track bugs?

Free Trial and Single User FreePack™ available at www.alexcorp.com

Alexsys Team® does! Alexsys Team 2 is a multi-user Team management system that provides a powerful yet easy way to manage all the members of your team and their tasks - including defect tracking. Use Team right out of the box or tailor it to your needs.

Track all your project tasks in one database so you can work together to get projects done.

- Quality Control / Compliance Tracking
- Project Management
- End User Accessible Service Desk Portal
- Bugs and Features
- Action Items
- Sales and Marketing
- Help Desk

Native Smart Card Login Support including Government and DOD

New in Team 2.11

- Full Windows 7 Support
- Windows Single Sign-on
- System Audit Log
- Trend Analysis
- Alternate Display Fields for Data Normalization
- Lookup Table Filters
- XML Export
- Network Optimized for Enterprise Deployment

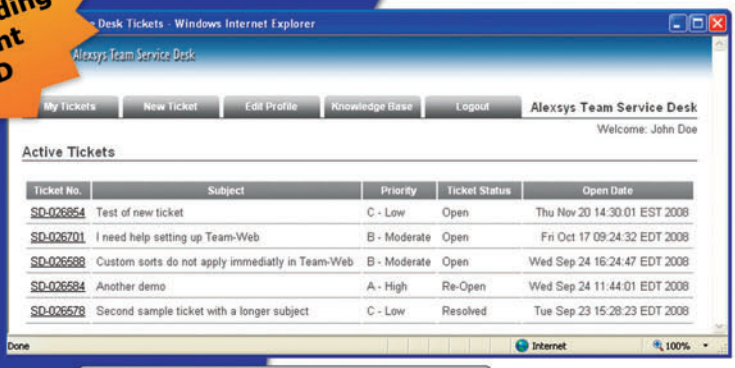
Service Desk Features

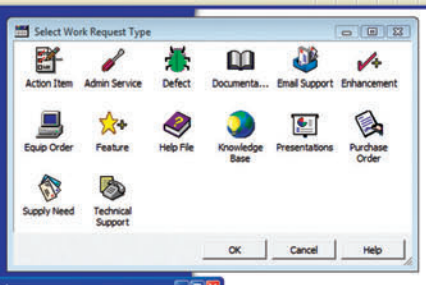
- Fully Secure
- Unlimited Users Self Registered or Active Directory
- Integrated into Your Web Site
- Fast/AJAX Dynamic Content
- Unlimited Service Desks
- Visual Service Desk Builder

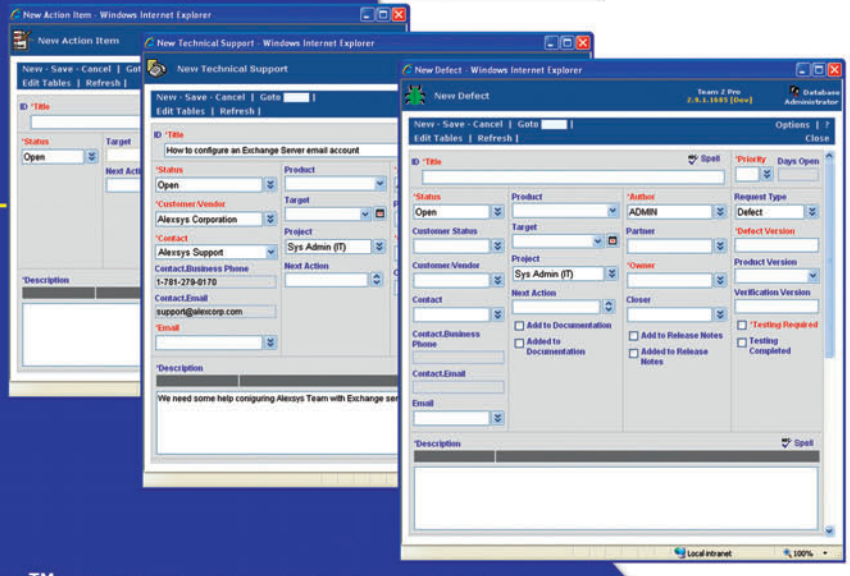
Team 2 Features

- Windows and Web Clients
- Multiple Work Request Forms
- Customizable Database
- Point and Click Workflows
- Role Based Security
- Clear Text Database
- Project Trees
- Time Recording
- Notifications and Escalations
- Outlook Integration

ZDNet Editors' Pick







Free Trial and Single User FreePack™ available at www.alexcorp.com.
FreePack™ includes a free single user Team Pro and Team-Web license.
Need more help? Give us a call at 1-888-880-ALEX (2539).

Team 2 works with its own standard database, while Team Pro works with Microsoft SQL, MySQL, and Oracle Servers.
Team 2 works with Windows 7/2008/2003/Vista/XP.
Team-Web works with Internet Explorer, Firefox, Netscape, Safari, and Chrome.

```
var dataProtectionProvider =
  Windows.Security.Cryptography.DataProtection.DataProtectionProvider(
  "LOCAL=user");
```

The DataProtectionProvider constructor (for encryption) takes a string argument that determines what the Data Protection is associated with. In this case, I'm encrypting content to the local user. Instead of setting it to the local user, I could set it to the machine, a set of Web credentials, an Active Directory security principle or a few other options. You can find a list of protection description options at the Dev Center topic, "Protection Descriptors" (bit.ly/QONGdG). Which protection descriptor you use depends on your app's requirements. At this point, the Data Protection Provider is ready to encrypt the data, but the data needs a slight change. Encryption algorithms work with buffers, not JSON, so the next step is to cast healthItem as a buffer:

```
var buffer =
  Windows.Security.Cryptography.CryptographicBuffer.convertStringToBinary(
    JSON.stringify(healthItem),
    Windows.Security.Cryptography.BinaryStringEncoding.utf8);
```

CryptographicBuffer has many objects and methods to work with buffers used in encryption and decryption. The first of these methods is convertStringToBinary, which takes a string (in this case, the string version of the JSON object) and converts it to an encoded buffer. The encoding used is set with the Windows.Security.Cryptography.BinaryStringEncoding object. In this example, I use UTF8 as the encoding for my data. The convertStringToBinary method returns a buffer based on the string data and the encoding specified. With the buffer ready to be encrypted and the Data Protection Provider instantiated, I'm ready to call the protectAsync method to encrypt the buffer:

```
dataProtectionProvider.protectAsync(buffer).then(
  function (encryptedBuffer) {
     SaveBufferToFile(encryptedBuffer);
});
```

The encryptedBuffer argument is the output of the protectAsync method and contains the encrypted version of the buffer. In other words, this is the encrypted data ready for storage. From here, encryptedBuffer is passed to the SaveBufferToFile method, which writes the encrypted data to a file in the app's local folder.

Encryption for healthItem boils down to three lines of code:

1. Instantiate the Data Protection Provider.
2. Convert the data to a buffer.
3. Call protectAsync to encrypt the data.

Decrypting data is just as simple. The only changes are to use an empty constructor for the DataProtectionProvider and use the unprotectAsync method instead of the protectAsync method. The GetBufferFromFile method loads the encryptedBuffer variable from the file created in the SaveBufferToFile method:

```
function btnLoadItem_Click(args) {
  var dataProtectionProvider =
    Windows.Security.Cryptography.DataProtection.DataProtectionProvider();

  var encryptedBuffer = GetBufferFromFile();

  dataProtectionProvider.unprotectAsync(encryptedBuffer).then(
    function (decryptedBuffer) {
      // TODO: Work with decrypted data
    });
}
```

Can developers use encryption with non-WinRT JavaScript? Yes! Is it as easy as three lines of code that provide excellent data protection? No! There are numerous challenges to encryption best practices in the browser, such as keeping the secret key a secret, as well as managing the file size of the algorithms necessary to have quality encryption. The WinRT Data Protection API as well as the other cryptography tools provided in the Windows.Security.Cryptography namespace make protecting your data simple. Using the security features of the Windows Runtime, developers can store sensitive data in their Windows Store app with confidence while keeping their cryptographic keys easy to manage.

Local vs. Web Contexts

Web developer says: Web apps execute external script references in the same origin of the application that calls the scripts.
Windows 8 developer says: Windows Store apps separate the local app package from external script references.

Web 2.0 has trained developers that content can come from your site, someone else's site (via mashup) or user interaction. On the Web, content is a virtual free-for-all, with developers consuming script references and API data from third parties. Content delivery networks (CDNs) and online services such as Bing Maps take away the overhead of managing code libraries or big data repositories, allowing Web applications to easily snap in functionality. Lower overhead is a good thing, but with this benefit comes some risk.

As an example, imagine one of Contoso's partners in the health-software industry is Litware Inc. Litware is releasing a new Exercise API and has provided Contoso Health developers with keys to consume a daily exercise data feed. If Contoso Health were a Web application, the development team could implement the Exercise API using a script reference like the following:

```
<script src="https://api.litware.com/devkey/exercise.js"></script>
```

Developers at Contoso trust Litware to provide great content and know it has great security practices. Unfortunately, Litware's servers were compromised by a disgruntled developer and exercise.js was altered to have a startup script that displays a pop-up with a message saying, "Contoso Health needs to run maintenance; please download the following maintenance application." The user, thinking this message is legitimate, was just tricked into downloading malware. Contoso's developers were baffled—Litware uses great validation, so how could this breach have happened?

On the Web, scripts referenced in the manner just described execute with the same origin as a script on the same site. That means

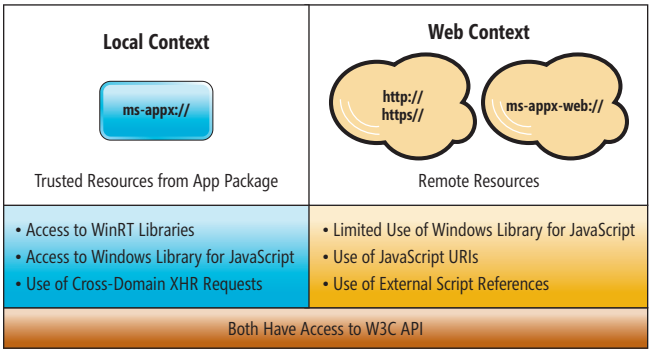


Figure 2 **Local vs. Web Context Features (Mashed from "Features and Restrictions by Context" [bit.ly/NZUyWt] and "Secure Development with HTML5" [bit.ly/JOoMOS])**

Figure 3 **Schemes with Context Examples**

Scheme	Content Location	Context	Example	When Used
ms-appx://	App package	Local	`<iframe src="ms-appx:///1.html" ></iframe>`	Load content into an iframe that needs to access the Windows Runtime or the full Windows JavaScript API.
ms-appx-web://	App package	Web	`<iframe src="ms-appx-web:///2.html" ></iframe>`	Use content from a remote source as part of your Windows Store app interface, such as displaying a mapping widget or search results.
http://	Remote	Web	`<iframe src="http://host/3.html" ></iframe>`	Reference remote content such as a Web page or script file on another server.

exercise.js (running as JavaScript) has unquestioned access to the DOM tree as well as any script object. As illustrated earlier, this can lead to serious security issues. To mitigate this risk, Windows 8 breaks app resources into two contexts, as illustrated in **Figure 2**.

The local context can access the Windows Runtime as well as any resource included in the app package (such as HTML, script, CSS and app data stored in the app state directories) but can't access remote HTML, JavaScript or CSS (as in the earlier exercise.js example). The top-level app in Windows 8 always runs in the local context. In **Figure 2**, ms-appx:// is used to resolve content in the local context. This scheme is used to reference content in the app package running within the local context. Often a third slash follows (ms-appx:///) to reference the package's full name. For Web developers, this approach is similar to using the file:// protocol, where a third slash references the local file system (file:/// assumes file://END USER's COMPUTER/ instead of file://REMOTE COMPUTER/).

The Web context allows developers to bring remote content into their Windows Store app through an iframe. Just like iframes in a Web browser, the content executing in the iframe is restricted from accessing resources outside of it, such as Windows Runtime and some features of Windows Library for JavaScript. (You can find a complete listing at bit.ly/PoQV0j.) The purpose of the Web context is to allow developers to reference third-party APIs such as Bing Maps or pull a library from a CDN into their app.

Using http:// or https:// as the source of an iframe automatically casts the contents of the iframe into the Web context. An iframe can also be a resource in the app package when you're using ms-appx or ms-appx-web. When the source of an iframe references the ms-appx:// scheme, the iframe's content runs in the local context. This allows developers to embed app package resources into an iframe while still having access to the features of the local context (such as Windows Runtime, Windows JavaScript API and so on). Another scheme available is ms-appx-web://, which allows local app package content to run in the Web context. This scheme is useful when you need to embed remote content within your markup, such as adding a Bing Search result (from the Bing Search API) of local hospitals based on the user's location in the Contoso Health app. As a side note, whenever iframes are mentioned with HTML5, remember that you can use the sandbox attribute as extra protection for your app by limiting script execution of the content inside the iframe. You can find more information about the sandbox attribute at bit.ly/Ppbo1a.

Figure 3 shows the various schemes used in the local and Web contexts along with examples of their use.

Which context an iframe belongs to is based on how the content within it is referenced. In other words, the scheme determines the context. You can find more information about the schemes used in Windows 8 at bit.ly/SS711o.

Remember the Litware hack scenario that started this section? The Windows 8 separation of contexts will help constrain the cross-site scripting attack to the Web context where it doesn't have access to either Windows Runtime or the app data for Contoso Health. In the Web context, modifying the local context isn't an option. Communication between the contexts is possible, but you have control over what type of communication occurs.

Communicating Between Contexts

How does the top-level document communicate with an iframe running in the Web context? Using the postMessage features of HTML5, Windows Store apps can pass data between contexts. This allows developers to structure how the two origins communicate and to allow only known good providers (the allow list again) through to the local context. Pages that need to run in the Web context are referenced using an iframe with the src attribute set to http://, https:// or ms-appx-web://.

For the Contoso Health app, the system pulls fitness tips from the Litware Exercise API. Contoso Health's development team has built the litwareHelper.html page, which is used to communicate with the Exercise API via the jQuery $ajax object. Because of the remote resource (exercise.js), litwareHelper.html needs to execute in the Web context, which means that it needs to run within an iframe. Setting up the iframe isn't different than in any other Web application except for how the page is referenced. Because the litwareHelper.html page is part of the local app package but needs to run in the Web context, you load it using ms-appx-web:

```
<iframe id="litwareHelperFrame" src="ms-appx-web:///litwareHelper.html"></iframe>
```

The development team adds the following function to the local context page that sends the data request to the Web context page:

```
function btnGetFitTips_Click() {
  var msg = {
    term: document.getElementById("txtExerciseSearchTerm").value,
    itemCount: 25  }
  var msgData = JSON.stringify(msg);
  var domain = "ms-appx-web://" + document.location.host;
  try {
    var iframe = document.getElementById("litwareHelperFrame");
    iframe.contentWindow.postMessage(msgData, domain);
  }
  catch (ex) {
    document.getElementById("output").innerText = "Error has occurred!";
  }
}
```

This function first builds the msg object, which is what the top-level document (always running in the local context) sends to the iframe (running as the Web context) for processing. Data sent via postMessage can be a JavaScript type (such as string or number), blob or file but not a JSON object. The JSON object is converted to a string so it can be sent via postMessage. After getting a reference to the litwareHelperFrame, the application calls postMessage to send the string version of the msg object to the frame and specifies the domain destination for the message. This is all Contoso Health needs to send data from the local context to a Web context page.

> Separating the local and Web contexts by default reduces the risk of accidentally executing code from an outside source.

Now the litwareHelper.html page needs to consume the message. For the litwareHelper.js code, you first add an event listener for the message event. This event is triggered when a message is received from another window or frame:

```
(function () {
'use strict';
  function initialize() {
    window.addEventListener('message', receiveMsg, false);
  }
  function receiveMsg(e) {
    ...
  }
  document.addEventListener("DOMContentLoaded", initialize, false);
})();
```

The receiveMsg method processes the message from the local context. The argument of receiveMsg is the data provided to the postMessage event (in this case, the msgData variable) along with the message target, message origin and a few other pieces of information, as shown in **Figure 4**.

The first step in receiveMsg checks the origin of the postMessage. This is a critical security check to ensure that the message is coming from where it's supposed to be originating. Remember that e.origin checks the domain and scheme of who sent the postMessage, which is why you're checking for ms-appx (the local context address). After gathering the JSON data from the Litware API, the app passes the results back to the window.parent using a postMessage command. Notice in receiveMsg that the domain is set to ms-appx. This is the "to" address of where the message is going and shows that the data is returning to the local context. Data from the iframe needs to be consumed by resources in the local context. The dev team adds the processResult function to consume the data from the Web context back into the local context:

```
function processResult(e) {
  if (e.origin === "ms-appx-web://" + document.location.host) {
    document.getElementById("output").innerText = e.data;
  }
}
```

Again, always check the origin of the message event to ensure that only data from approved locations (that is, locations that are registered in the allow list) is being processed. Notice that the origin is the Web context scheme: ms-appx-web in the processResult method. The switch between schemes can be a gotcha that developers can overlook and wonder where their message went during debugging.

Finally, to receive data from the Web context back to the local context page, you add an event handler for the message event. In the app.onactivated method, add the event listener to the window object:

```
window.addEventListener('message', processResult, false);
```

Separating the local and Web contexts by default reduces the risk of accidentally executing code from a source outside the Windows Store app. Using postMessage, developers can provide a communication channel between external script and the local scripts that compose an app.

Figure 4 **Processing with receiveMsg**

```
function receiveMsg(e) {
  if (e.origin === "ms-appx://" + document.location.host) {
    var output = null;
    var parameters = JSON.parse(e.data);
    var url = "https://api.litware-exercise.com/data/" + parameters.term +
      "/count/" + parameters.itemCount;
    var options = {
      dataType: "jsonp",
      jsonpCallback: "jsonpCallback",
      success: function (results) {
        output = JSON.stringify(results.items);
        window.parent.postMessage(output, "ms-appx://" + document.location.host);
      },
      error: function (ex) {
        output = ex;
      }
    };
    $.ajax(url, options);
  }
}
```

Web to Windows 8: New Tools for Old Habits

Web developers now have access to familiar tools and new tools they can use to build secure Windows Store apps. Using existing skills, such as HTML5 input validation, ensures the integrity of data entering the app. New tools such as the Data Protection API (new for Windows Runtime) protect user's confidential data with strong encryption that's simple to implement. Using postMessage allows apps to tap into the thousands of JavaScript libraries and legacy code on the Web while keeping users safe from unintended code injections. All these elements work together to bring something important that's often dismissed in JavaScript: security.

Windows 8 gives Web developers the chance to rethink some of their old habits. JavaScript is no longer a façade for the server, dismissed as an enhancement to usability and nothing more. JavaScript, the Windows Runtime and MSHTML provide the tools necessary to build security features into your Windows Store apps—no server necessary. As Web developers, we have a vast skillset to draw on, but we need to keep an eye on our old habits and turn them into opportunities to learn the new world of Windows 8. ■

Tim Kulp *leads the development team at FrontierMEDEX in Baltimore, Md. You can find Kulp on his blog at seccode.blogspot.com or follow him on Twitter at Twitter.com/seccode, where he talks code, security and the Baltimore foodie scene.*

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

Compatible with
Microsoft® Visual Studio® 2012

FREE TRIAL DOWNLOAD
INFRAGISTICS.COM/DOWNLOADS

GET A PULSE

On Mobile Business Intelligence

infragistics.com/ **EXPERIENCE**

INFRAGISTICS
DESIGN / DEVELOP / EXPERIENCE

Infragistics Sales US 800 231 8588 • Europe +44 (0) 800 298 9055 • India +91 80 4151 8042 • APAC (+61) 3 9982 4545

Speech-Enabling a Windows Phone 8 App with Voice Commands

F Avery Bishop

One recent evening I was running late for an after-work meeting with an old friend. I knew he was already driving to the rendezvous, so calling him would be out of the question. Nevertheless, as I dashed out of my office and ran toward my car, I grabbed my Windows Phone and held down the Start button. When I heard the "earcon" listening prompt, I said, "Text Robert Brown," and when the text app started up, I said, "Running late, leaving office now," followed by "send" to send the text message.

Without the speech features in the built-in texting app, I would've had to stop running and fumble around in frustration to send a text because I find the keypad hard to use with my fat fingers and the screen difficult to read while on the run. Using speech to text saved me time, frustration and no small amount of anxiety.

Windows Phone 8 offers these same speech features for developers to interact with their users through speech recognition and text-to-speech. These features support the two scenarios illustrated in my example: From anywhere on the phone, the user can say a command to launch an app and carry out an action with just one utterance; and once in the app, the phone carries on a dialog with the user by capturing commands or text from the speaker's spoken utterances and by audibly rendering text to the user for notification and feedback.

The first scenario is supported by a feature called voice commands. To enable this feature, the app provides a Voice Command Definition (VCD) file to specify a set of commands that the app is equipped to handle. When the app is launched by voice commands, it receives parameters in a query string such as the command name, parameter names and the recognized text that it can use to execute the command specified by the user. This first installment of a two-part article explains how to enable voice commands in your app on Windows Phone 8.

In the second installment I'll discuss in-app speech dialog. To support this, Windows Phone 8 provides an API for speech recognition and synthesis. This API includes a default UI for confirmation and disambiguation as well as default values for speech grammars, timeouts and other properties, making it possible to add speech recognition to an app with just a few lines of code. Similarly, the speech synthesis API (also known as text-to-speech, or TTS) is easy to code for simple scenarios; it also provides advanced features such as fine-tuned manipulation via the World Wide Web Consortium Speech Synthesis Markup Language (SSML) and switching between end-user voices already on the phone or downloaded from the marketplace. Stay tuned for a detailed exploration of this feature in the follow-up article.

This article discusses:

- Requirements
- Specifying voice commands
- Enabling voice commands
- Handling voice commands
- Updating phrase lists

Technologies discussed:

Windows Phone 8

Code download available at:

archive.msdn.microsoft.com/mag201211WP8Speech

To demonstrate these features, I've developed a simple app called Magic Memo. You can launch Magic Memo and execute a command by holding the Start button and then speaking a command when prompted. Inside the app, you can enter your memo using simple dictation or navigate within the app and execute commands using speech. Throughout this article, I'll explain the source code that implements these features.

Requirements for Using Speech Features in Apps

The Magic Memo app should work out of the box, assuming your development environment meets the hardware and software requirements for developing Windows Phone 8 apps and testing on the phone emulator. When this article went to press the requirements were as follows:

- 64-bit version of Windows 8 Pro or higher
- 4GB or more of RAM
- Second Level Address Translation supported by the BIOS
- Hyper-V installed and running
- Visual Studio 2012 Express for Windows Phone or higher

As always, it's best to check MSDN documentation for the latest requirements before attempting to develop and run your app.

Three other things to keep in mind when you develop your own app from scratch:

1. Ensure that the device microphone and speaker are working properly.
2. Add capabilities for speech recognition and microphone to the WpAppManifest.xml file either by checking the appropriate boxes in the properties editor or manually by including the following in the XML file:
```
<Capability Name="ID_CAP_SPEECH_RECOGNITION"/>
<Capability Name="ID_CAP_MICROPHONE"/>
```
3. When attempting speech recognition, you should catch the exception thrown when the user hasn't accepted the speech privacy policy. The GetNewMemoByVoice helper function in MainPage.xaml.cs in the accompanying sample code download gives an example of how to do this.

Figure 1 **Voice Command Definition File for the Magic Memo App**

```
<?xml version="1.0" encoding="utf-8"?>
<VoiceCommands xmlns="http://schemas.microsoft.com/voicecommands/1.0">
  <CommandSet xml:lang="en-us" Name="MagicMemoEnu">
    <!-- Command set for all US English commands-->
    <CommandPrefix>Magic Memo</CommandPrefix>
    <Example>enter a new memo</Example>

    <Command Name="newMemo">
      <Example>enter a new memo</Example>
      <ListenFor>Enter [a] [new] memo</ListenFor>
      <ListenFor>Make [a] [new] memo</ListenFor>
      <ListenFor>Start [a] [new] memo</ListenFor>
      <Feedback>Entering a new memo</Feedback>
      <Navigate />    <!-- Navigation defaults to Main page -->
    </Command>

    <Command Name="showOne">
      <Example>show memo number two</Example>
      <ListenFor>show [me] memo [number] {num} </ListenFor>
      <ListenFor>display memo [number] {num}</ListenFor>
      <Feedback>Showing memo number {num}</Feedback>
      <Navigate Target="/ViewMemos.xaml"/>
    </Command>

    <PhraseList Label="num">
      <Item> 1 </Item>
      <Item> 2 </Item>
      <Item> 3 </Item>
    </PhraseList>
  </CommandSet>

  <CommandSet xml:lang="ja-JP" Name="MagicMemoJa">
    <!-- Command set for all Japanese commands -->
    <CommandPrefix>マジック・メモ</CommandPrefix>
    <Example>新規メモ</Example>

    <Command Name="newMemo">
      <Example>新規メモ</Example>
      <ListenFor>新規メモ[を]</ListenFor>
      <ListenFor>新しいメモ</ListenFor>
      <Feedback>メモを言ってください</Feedback>
      <Navigate/>
    </Command>

    <Command Name="showOne">
      <Example>メモ１を表示</Example>
      <ListenFor>メモ{num}を表示[してください] </ListenFor>
      <Feedback>メモ{num}を表示します。 </Feedback>
      <Navigate Target="/ViewMemos.xaml"/>
    </Command>

    <PhraseList Label="num">
      <Item> 1 </Item>
      <Item> 2 </Item>
      <Item> 3 </Item>
    </PhraseList>
</CommandSet>
</VoiceCommands>
```

Inside the app, you can enter your memo using simple dictation or navigate within the app and execute commands using speech.

The Scenario

On any smartphone, a common scenario is to launch an app and execute a single command, optionally followed by more commands. Doing this manually requires several steps: finding the app, navigating to the right place, finding the button or menu item, tapping on that, and so on. Many users find this frustrating even after they've become accustomed to the steps.

For example, to display a saved memo—for instance, memo No. 12—in the Magic Memo sample app, the user must find and launch the app, tap on "View saved memos" and scroll down until the desired memo is displayed. Contrast this with the experience of using the Windows Phone 8 voice commands feature: The user holds the Start button and says "Magic Memo, show memo 12," after which the Magic Memo app is launched and the desired memo is displayed in a message box. Even for this simple command, there's a clear savings in user interaction.

There are three steps to implementing voice commands in an app and an optional fourth step for handling dynamic content. The following sections outline those steps.

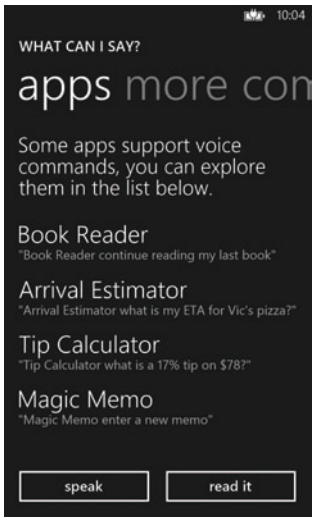


Figure 2 **Help Page Showing Voice Command Examples for Installed Apps**

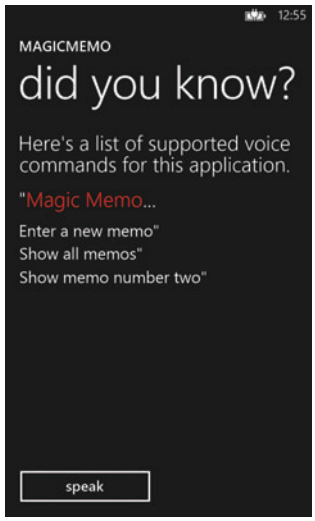


Figure 3 **Example Page for Magic Memo Voice Commands**

Specifying the User Commands to Recognize

The first step to implementing voice commands is to specify the commands to listen for in a VCD file. A VCD file is authored in a simple XML format consisting of a collection of CommandSet elements, each with Command child elements that contain the phrases to listen for. An example from the Magic Memo app is shown in **Figure 1**.

Following are guidelines to design a VCD file:

1. Keep the command prefix phonetically distinct from Windows Phone keywords. This will help to avoid confusing your app with a built-in phone feature. For U.S. English, the keywords are *call, dial, start, open, find, search, text, note* and *help*.
2. Make your command prefix a subset or a natural pronunciation of your app name rather than something completely different. This will avoid user confusion and reduce the chance of misrecognizing your app for some other app or feature.
3. Keep in mind that recognition requires an exact match on the command suffix. Thus, it's a good idea to keep the command prefix simple and easy to remember.
4. Give each command set a Name attribute so that you can access it in your code.
5. Keep ListenFor elements in different Command elements phonetically distinct from each other to reduce the chances of misrecognition.
6. Ensure that ListenFor elements in the same command are different ways to specify the same command. If ListenFor elements in a command correspond to more than one action, split them into separate commands. This will make it easier to handle the commands in your app.
7. Keep in mind the limits: 100 Command elements in a command set; 10 ListenFor entries in a command; 50 total PhraseList elements; and 2,000 total PhraseList items across all PhraseLists.
8. Keep in mind that recognition on PhraseList elements requires an exact match, not a subset. Thus, to recognize both "Star Wars" and "Star Wars Episode One," you should include both as PhraseList elements.

In my example there are two CommandSet elements, each with different xml:lang and Name attributes. There can be only one CommandSet per xml:lang value. The Name attributes must also be unique but are restricted only by the Name attribute's value specification. Though optional, it's highly recommended that you include a Name attribute because you'll need it to access the CommandSet from your app code to implement step 4. Also note that only one CommandSet is active for your app at one time, namely the one whose xml:lang attribute exactly matches that of the current global speech recognizer, as set by the user in SETTINGS/speech. You should include CommandSets for any languages you expect your users to require in their market.

The next thing to notice is the CommandPrefix element. Think of this as an alias the user can say to invoke your app. This is useful if your app name has nonstandard spelling or nonpronounceable characters, such as Mag1c or gr00ve. Remember that this word or phrase has to be something that the speech recognition engine can recognize and also that it must be phonetically distinct from Windows Phone built-in keywords.

You'll note there are Example elements as children of both the CommandSet element and of the Command element. The Example under CommandSet is a general example for your app that will show up on the system help What can I say? screen as shown in **Figure 2**. In contrast, the Example element under a Command is specific to that Command. This Example shows up on a system help page (see **Figure 3**) that's displayed when the user taps on the app name in the help page shown in **Figure 2**.

Speaking of which, each Command child element within a CommandSet corresponds to an action to take in the app once launched. There may be multiple ListenFor elements in a Command, but they should all be different ways of telling the app to carry out the action (command) of which they are a child.

Note also that the text in a ListenFor element has two special constructs. Square braces around text means the text is optional—that is, the user's utterance can be recognized with or without the enclosed text. Curly braces contain a label that references a PhraseList element. In the U.S. English example in **Figure 1**, the first

Figure 4 **Initializing the VCD file from Within the App**

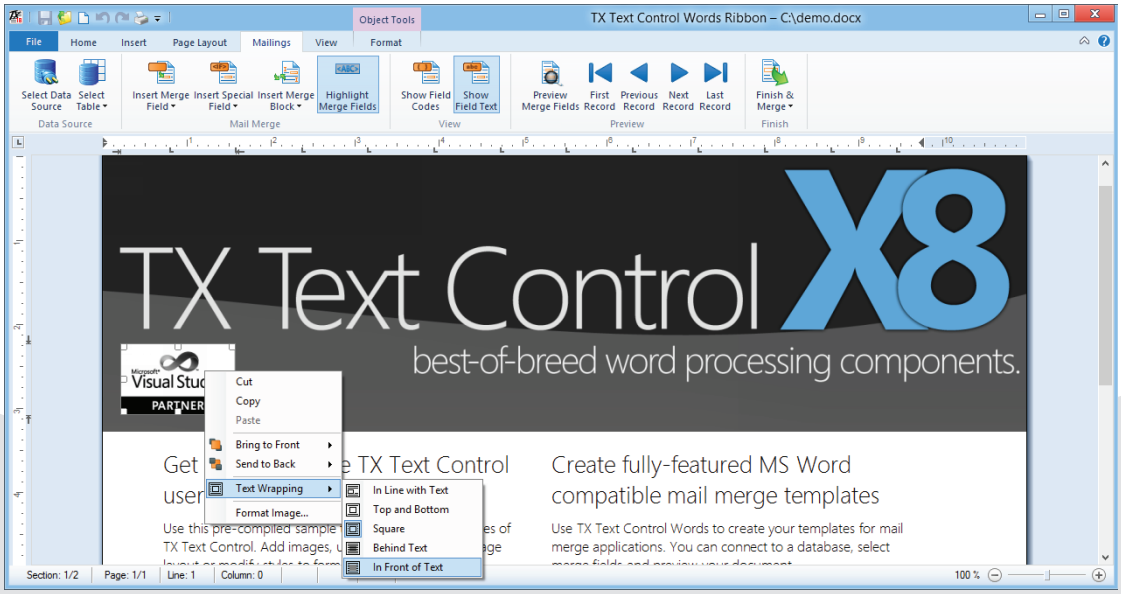
```
using Windows.Phone.Speech.VoiceCommands;

// ...

// Standard boilerplate method in the App class in App.xaml.cs
private async void Application_Launching(object sender, LaunchingEventArgs e)
{
  try // try block recommended to detect compilation errors in VCD file
  {
    await VoiceCommandService.InstallCommandSetsFromFileAsync(
      new Uri("ms-appx:///MagicMemoVCD.xml"));
  }
  catch (Exception ex)
  {
    // Handle exception
  }
}
```

WINDOWS FORMS | WPF | ASP.NET | ACTIVEX

WORD PROCESSING COMPONENTS

Ⓘ Rich Text Editing

Integrate professional rich text editing into your .NET based applications.



PDF Reflow - Open PDF Documents

Load, view, modify and convert Adobe PDF documents and reuse formatted text. Search and analyze documents such as invoices or delivery notes.

Free License

Download our 100% free Express version.

Spell Checking

Add the fastest spell checking engine to your Windows Forms, WPF or ASP.NET applications.

Reporting Redefined

Build MS Word compatible mail merge applications with Master-Detail views.

TX TEXT CONTROL® word processing components

Microsoft® Visual Studio® PARTNER

Word Processing Components for Windows Forms, WPF & ASP.NET

WWW.TEXTCONTROL.COM

US +1 877-462-4772 (toll-free)
EU +49 421-4270671-0

ListenFor under the "showOne" command has a label {num} referencing the phrase list below it. You can think of this as a slot that can be filled with any of the phrases in the referenced list, in this case numbers.

What happens when a command is recognized in the user's utterance? The phone's global speech recognizer will launch the app at the page specified in the Target attribute of the Navigate element under the corresponding Command, as explained later in step 3. But first, I'll discuss step 2.

Enabling Voice Commands

Having included the VCD file in your installation package, step 2 is to register the file so that Windows Phone 8 can include the app's commands in the system grammar. You do this by calling a static method InstallCommandSetsFromFileAsync on the VoiceCommandService class, as shown in **Figure 4**. Most apps will make this call on first run, but of course it can be done at any time. The implementation of VoiceCommandService is smart enough to do nothing on subsequent calls if there has been no change in the file, so don't worry about the fact that it's called on each launch of the app.

As the method name InstallCommandSetsFromFileAsync implies, the operational unit in the VCD file is a CommandSet element rather than the file itself. The call to this method inspects and validates all of the command sets contained in the file, but it installs only the one whose xml:lang attribute matches exactly that of the global speech engine. If the user switches the global recognition language to one that matches the xml:lang of a different CommandSet in your VCD, that CommandSet will be loaded and activated.

Handling a Voice Command

Now I'll discuss step 3. When the global speech recognizer recognizes the command prefix and a command from your app, it launches the app at the page specified in the Target attribute of the Navigate element, using your default task target (usually MainPage.xaml for Silverlight apps) if no Target is specified. It also appends to the query string key/value pairs for the Command name and PhraseList values. For example, if the recognized phrase is, "Magic Memo show memo number three," the query string might look something like the following (the actual string may vary by implementation or version):

```
"/ViewMemos.xaml?voiceCommandName=show&num=3&reco=show%20memo%20number%20three"
```

Fortunately, you don't have to parse the query string and dig out the parameters yourself because they're available on the NavigationContext object's QueryString collection. The app can use this data to determine whether it was launched by voice command—and, if so, handle the command appropriately (for example, in the page's Loaded handler). **Figure 5** shows an example from the Magic Memo app for the ViewMemos.xaml page.

> When the global speech recognizer recognizes the command prefix and a command from your app, it launches the app at the page specified in the Target attribute of the Navigate element.

Figure 5 **Handling Voice Commands in an App**

```
// Takes appropriate action if the application was launched by voice command.
private void ViewMemosPage_Loaded(object sender, RoutedEventArgs e)
{
  // Other code omitted

  // Handle the case where the page was launched by Voice Command
  if (this.NavigationContext.QueryString != null
    && this.NavigationContext.QueryString.ContainsKey("voiceCommandName"))
  {
    // Page was launched by Voice Command
    string commandName =
      NavigationContext.QueryString["voiceCommandName"];
    string spokenNumber = "";

    if (commandName == "showOne" &&
      this.NavigationContext.QueryString.TryGetValue("num", out spokenNumber))
    {
      // Command was "Show memo number 'num'"
      int index = -1;
      if (int.TryParse(spokenNumber, out index) &&
        index <= memoList.Count && index > 0)
      { // Display the specified memo
        this.Dispatcher.BeginInvoke(delegate
          { MessageBox.Show(String.Format(
          "Memo {0}: \"{1}\"", index, memoList[index - 1])); });
      }
    }
    // Note: no need for an "else" block because if launched by another VoiceCommand
    // then commandName="showAll" and page is shown
  }
}
```

Because there's more than one way to navigate to any page, the code in **Figure 5** first checks for the presence of the voiceCommandName key in the query string to determine if the user launched the app by a voice command. If so, it verifies the command name and gets the value of the PhraseList parameter num, which is the number of the memo the user wishes to see. This page has only two voice commands and the processing is simple, but a page that can be launched by many voice commands would use something like a switch block on the commandName to decide what action to take.

The PhraseList in this example is also simple; it's just a series of numbers, one for each stored memo. You can envision more sophisticated scenarios, however, requiring phrase lists that are populated dynamically—for example, from data on a Web site. The optional step 4 mentioned earlier addresses how to implement PhraseLists for these scenarios. I'll discuss it next.

Updating Phrase Lists from Your App

You may have noticed a problem with the VCD file in **Figure 1**: The "num" PhraseList defined statically in the VCD supports recognition up to three items, but eventually there are likely to be many more than three memos stored in the app's isolated storage. For use cases where the phrase list changes over time, there's a way to update the

Figure 6 **Updating the Installed Phrase Lists Dynamically**

```
// Updates the "num" PhraseList to have the same number of
// entries as the number of saved memos; this supports
// "Magic Memo show memo 5" if there are five or more memos saved
private async void UpdateNumberPhraseList(string phraseList,
  int newLimit, string commandSetName)
{
  // Helper function that sets string array to {"1", "2", etc.}
  List<string> positiveIntegers =
    Utilities.GetStringListOfPositiveIntegers(Math.Max(1, newLimit));

  try
  {
    VoiceCommandSet vcs = null;

    if (VoiceCommandService.InstalledCommandSets.TryGetValue(
      commandSetName, out vcs))
    {
      // Update "num" phrase list to the new numbers
      await vcs.UpdatePhraseListAsync(phraseList, positiveIntegers);
    }
  }
  catch (Exception ex)
  {
    this.Dispatcher.BeginInvoke(delegate
      { MessageBox.Show("Exception in UpdateNumberPhraseList " + ex.Message); }
    );
  }
}
```

phrase list dynamically from within the app, as shown in **Figure 6**. This is especially useful for apps that need to recognize against dynamic lists such as downloaded movies, favorite restaurants or points of interest near the phone's current location.

Although the Magic Memo app doesn't demonstrate it, dynamically updated phrase lists are a perfect candidate for updating in a user agent because the updating can happen behind the scenes, even when the app isn't running.

And there you have it: four steps to enabling voice commands in your app. Try it out with the Magic Memo sample app. Remember that you need to run it once normally to load the VCD file, but after that you can say things such as the following to launch the app and take you right to a page and carry out the command:

- Magic Memo, enter a new memo
- Magic Memo, show all memos
- Magic Memo, show memo number four

Next Up: In-App Dialog

Implementing voice commands as I've discussed in this article is the first step to letting your users interact with your app on Windows Phone 8 just like they can with built-in apps such as Text, Find and Call.

The second step is to provide in-app dialog, in which the user speaks to your app after it's launched to record text or execute commands, and receives audio feedback as spoken text. I'll delve into that topic in Part 2, so stay tuned. ■

F Avery Bishop *has been working in software development for more than 20 years, 12 years of that at Microsoft, where he's a program manager for the speech platform. He has published numerous articles on natural language support in applications including topics such as complex script support, multilingual applications and speech recognition.*

Thanks to the following technical experts for reviewing this article: *Robert Brown, Victor Chang, Jay Waltmunson and Travis Wilson*

msdn magazine

STATEMENT OF OWNERSHIP, MANAGEMENT AND CIRCULATION

1. Publication Title: MSDN Magazine
2. Publication Number: 1528-4859
3. Filing Date: 9/28/12
4. Frequency of Issue: Monthly with a special issue in October
5. Number of Issues Published Annually: 13
6. Annual Subscription Price: US $35, International $60
7. Complete Mailing Address of Known Office of Publication: 9201 Oakdale Ave., Ste. 101, Chatsworth, CA 91311
8. Complete Mailing Address of the Headquarters of General Business Offices of the Publisher: Same as above.
9. Full Name and Complete Mailing Address of Publisher, Editor, and Managing Editor:
   Henry Allain, President, 4 Venture, Ste. 150, Irvine, CA 92618
   Matt N. Morollo, VP/Group Publisher, 600 Worcester Rd., Ste. 204, Framingham, MA 01702
   Michael Desmond, Editor-in-Chief, 600 Worcester Rd., Ste. 204, Framingham, MA 01702
   Wendy Hernandez, Group Managing Editor, 4 Venture, Ste. 150, Irvine, CA 92618
10. Owner(s): 1105 Media, Inc. dba: 101 Communications LLC, 9201 Oakdale Ave, Ste. 101, Chatsworth, CA 91311. Listing of shareholders in 1105 Media, Inc.
11. Known Bondholders, Mortgagees, and Other Security Holders Owning or Holding 1 Percent or more of the Total Amount of Bonds, Mortgages or Other Securities:
    Nautic Partners V, L.P., 50 Kennedy Plaza, 12th Flr., Providence, RI 02903
    Kennedy Plaza Partners III, LLC, 50 Kennedy Plaza, 12th Flr., Providence, RI 02903
    Alta Communications 1X, L.P., 1X-B, L.P., Assoc., LLC, 28 State St., Ste. 1801, Boston, MA 02109
12. The tax status has not changed during the preceding 12 months.
13. Publication Title: MSDN Magazine
14. Issue date for Circulation Data Below: September 2012
15. Extent & Nature of Circulation:

	Average No. Copies Each Month During Preceding 12 Months	No. Copies of Single Issue Published Nearest to Filing Date
a. Total Number of Copies (Net Press Run)	83,056	82,030
b. Legitimate Paid/and or Requested Distribution		
1. Outside County Paid/Requested Mail Subscriptions Stated on PS Form 3541	65,402	52,218
2. In-County Paid/Requested Mail Subscriptions Stated on PS Form 3541	0	0
3. Sales Through Dealers and Carriers, Street Vendors, Counter Sales, and Other Paid or Requested Distribution Outside USPS®	5,591	5,367
4. Requested Copies Distributed by Other Mail Classes Through the USPS	0	0
c. Total Paid and/or Requested Circulation	70,993	57,585
d. Nonrequested Distribution		
1. Outside County Nonrequested Copies Stated on PS Form 3541	9,789	23,241
2. In-County Nonrequested Copies Distribution Stated on PS Form 3541	0	0
3. Nonrequested Copies Distribution Through the USPS by Other Classes of Mail	0	0
4. Nonrequested Copies Distributed Outside the Mail	2,096	1,037
e. Total Nonrequested Distribution	11,885	24,278
f. Total Distribution	82,878	81,863
g. Copies not Distributed	178	167
h. Total	83,056	82,030
i. Percent paid and/or Requested Circulation	85.66%	70.34%

16. ☒ **Total Circulation includes elections copies. Report circulation on PS Form 3526X worksheet.**
17. Publication of Statement of Ownership for a Requester Publication is required and will be printed in the November 2012 issue of this publication.
18. I certify that all information furnished on this form is true and complete:
    Jenny Hernandez-Asandas, Director, Print and Online Production

If you are using PS Form 3526R and claiming electronic copies complete below:

a. Requested and Paid Electronic Copies	12,941	9,776
b. Total Requested and Paid Print Copies (Line 15C) + Paid Electronic Copies	83,934	67,361
c. Total Requested Copy Distribution (Line 15F) + Paid Electronic Copies	95,819	91,639
d. Percent Paid and/or Requested Circulation (Both Print & Electronic Copies)	87.60%	73.51%

☒ **I Certify that 50% of all my distributed copies (Electronic & Print) are legitimate requests.**

Managing Memory in Windows Store Apps, Part 2

Chipalo Street and Dan Taylor

In the Windows 8 special edition of *MSDN Magazine*, the first article in this series discussed how memory leaks occur, why they slow down your app and degrade the overall system experience, general ways to avoid leaks, and specific issues that have been found to be problematic in JavaScript apps (see "Managing Memory in Windows Store Apps," msdn.microsoft.com/magazine/jj651575). Now we'll look at memory leaks in the context of C#, Visual Basic and C++ apps. We'll analyze some basic ways leaks have occurred in past generations of apps, and how Windows 8 technologies help you avoid these situations. With this foundation, we'll move to more complex scenarios that can cause your app to leak memory. Let's get to it!

This article discusses:

- Leaks caused by reference cycles
- Long-lived event sources
- Event handlers that cross language boundaries
- Analyzing memory growth with PerfView

Technologies discussed:

Windows 8, C#, Visual Basic, C++

Simple Cycles

In the past, many leaks were caused by reference cycles. Objects involved in the cycle would always have an active reference even if the objects in the cycle could never be reached. The active reference would keep the objects alive forever, and if a program created these cycles frequently, it would continue to leak memory over time.

A reference cycle can occur for multiple reasons. The most obvious is when objects explicitly reference each other. For example, the following code results in the picture in **Figure 1**:

```
Foo a = new Foo();
Bar b = new Bar();
a.barVar = b;
b.fooVar = a;
```

Thankfully, in garbage-collected languages such as C#, JavaScript and Visual Basic, this kind of circular reference will be automatically cleaned up once the variables are no longer needed.

C++/CX, in contrast, doesn't use garbage collection. Instead, it relies on reference counts to perform memory management. This means that objects will be reclaimed by the system only when they have zero active references. In these languages, the cycles between these objects would force A and B to live forever because they would never have zero references. Even worse, everything referenced by A and B would live forever as well. This is a simplified example that can be easily avoided when writing basic programs; however, complex programs can create cycles that involve multiple objects chaining together in non-obvious ways. Let's take a look at some examples.

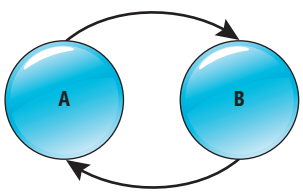


Figure 1 **A Circular Reference**

Figure 2 **Causing a Circular Reference with an Event Handler**

```
<MainPage x:Class="App.MainPage" ...>
  ...
  <TextBlock x:Name="displayTextBlock" ... />
  <Button x:Name="myButton" Click="ButtonClick" ... />
  ...
</MainPage>

public sealed partial class MainPage : Page
{
  ...
  private void ButtonClick(object sender, RoutedEventArgs e)
  {
    DateTime currentTime = DateTime.Now;
    this.displayTextBlock.Text = currentTime.ToString();
  }
  ...
}
```

Cycles with Event Handlers

As discussed in the earlier article, event handlers are an extremely common way for circular references to be created. **Figure 2** shows how this might occur.

Here we've simply added a Button and TextBlock to a Page. We've also set up an event handler, defined on the Page class, for the Button's Click event. This handler updates the text in the TextBlock to show the current time whenever the Button is clicked. As you'll see, even this simple example has a circular reference.

The Button and TextBlock are children of the page and therefore the page must have a reference to them, as shown in the top diagram in **Figure 3**.

In the bottom diagram, another reference is created by the registration of the event handler, which is defined on the Page class.

The event source (Button) has a strong reference to the event handler, a delegate method, so that the source can call the event handler when the event is fired. Let's call this delegate a strong delegate because the reference from it is strong.

> Creating event handers is an extremely common scenario and Microsoft doesn't want this to cause leaks in your app regardless of the language you use.

We now have a circular reference. Once the user navigates away from the page, the garbage collector (GC) is smart enough to reclaim the cycle between the Page and Button. These types of circular references will be automatically cleaned up when they're no longer needed if you're writing apps in JavaScript, C# or Visual Basic. As we noted earlier, however, C++/CX is a ref-counted language, which means objects are automatically deleted only when their reference count drops to zero. Here, the strong references created would force the Page and Button to live forever because they would never have zero reference counts. Even worse, all of the items contained by the Page (potentially a very large element tree) would live forever as well because the Page holds references to all of these objects.

Of course, creating event handers is an extremely common scenario and Microsoft doesn't want this to cause leaks in your app regardless of the language you use. For that reason the XAML compiler makes the reference from the delegate to the event listener a weak reference. You can think of this as a weak delegate because the reference from the delegate is a weak reference.

> The weak delegate ensures that the page isn't kept alive by the reference from the delegate to the page.

The weak delegate ensures that the page isn't kept alive by the reference from the delegate to the page. The weak reference will not count against the page's reference count and thus will allow it to be destroyed once all other references drop to zero. Subsequently, the Button, TextBlock and anything else referenced by the page will be destroyed as well.

Long-Lived Event Sources

Sometimes an object with a long lifetime defines events. We refer to these events as long-lived because the events share the lifetime of the object that defines them. These long-lived events hold references to all registered handlers. This forces the handlers, and the objects targeted by the handlers, to stay alive as long as the long-lived event source.

In the "Event Handler" section of the previous memory leak article, we analyzed one example of this. Each page in an app registers for the app window's SizeChangedEvent. The reference from the window's SizeChangedEvent to the event handler on the page will keep each instance of Page alive as long as the app's window is around. All of the pages that have been navigated to remain alive even though only one of them is in view. This leak is easily fixed by unregistering each page's SizeChangedEvent handler when the user navigates away from the page.

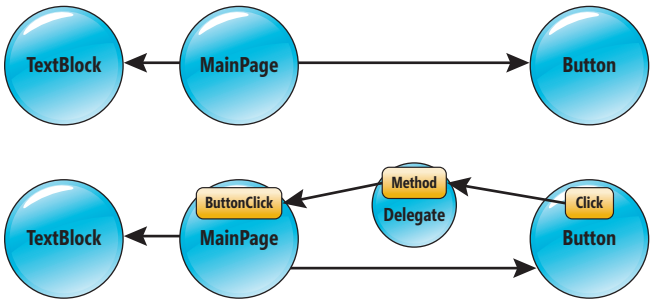


Figure 3 **A Circular Reference Related to the Event Handler**

In that example, it's clear when the page is no longer needed and the developer is able to unregister the event handler from the page. Unfortunately it's not always easy to reason about an object's lifetime. Consider simulating a "weak delegate" in C# or Visual Basic if you find leaks caused by long-lived events holding on to objects via event handlers. (See "Simulating 'Weak Delegates' in the CLR" at bit.ly/SUqw72.) The weak delegate pattern places an intermediate object between the event source and the event handler. Use a strong reference from the event source to the intermediate object and a weak reference from the intermediate object to the event handler, as shown in **Figure 4**.

In the top diagram in **Figure 4**, LongLivedObject exposes EventA and ShortLivedObject registers EventAHandler to handle the event. LongLivedObject has a much greater life span than ShortLivedObject and the strong reference between EventA and EventAHandler will keep ShortLivedObject alive as long as LongLivedObject. Placing an IntermediateObject between LongLivedObject and ShortLivedObject (as shown in the bottom diagram) allows IntermediateObject to be leaked instead of ShortLivedObject. This is a much smaller leak because the IntermediateObject needs to expose only one function, while ShortLivedObject may contain large data structures or a complex visual tree.

Let's take a look at how a weak delegate could be implemented in code. An event many classes may want to register for is DisplayProperties.OrientationChanged. DisplayProperties is actually a static class, so the OrientationChanged event will be around forever. The event will hold a reference to each object you use to listen to the event. In the example depicted in **Figure 5** and **Figure 6**, the class LargeClass uses the weak delegate pattern to ensure that the OrientationChanged event holds a strong reference only to an intermediate class when an event handler is registered. The intermediate class then calls the method, defined on LargeClass, which actually does the necessary work when the OrientationChanged event is fired.

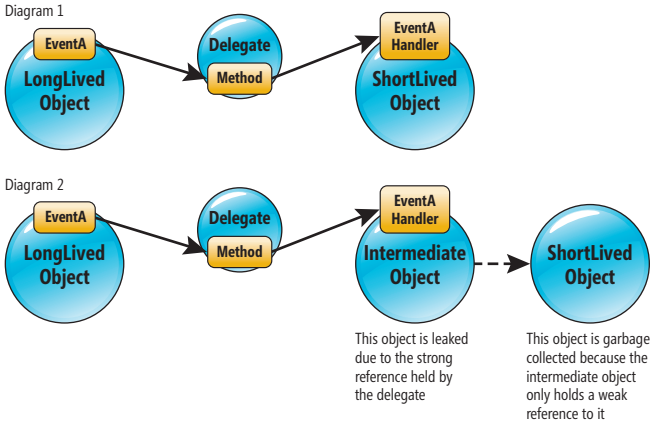


Figure 4 **Using an Intermediate Object Between the Event Source and the Event Listener**

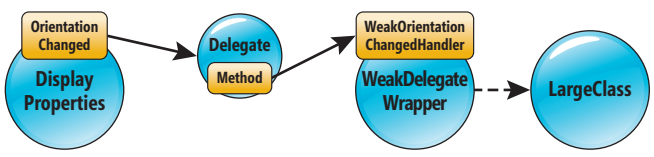


Figure 5 **The Weak Delegate Pattern**

Lambdas

Many people find it easier to implement event handlers with a lambda—or inline function—instead of a method. Let's convert the example from **Figure 2** to do exactly that (see **Figure 7**).

Figure 6 **Implementing a Weak Delegate**

```
public class LargeClass
{

  public LargeClass()
  {
    // Create the intermediate object
    WeakDelegateWrapper wrapper = new WeakDelegateWrapper(this);

    // Register the handler on the intermediate with
    // DisplayProperties.OrientationChanged instead of
    // the handler on LargeClass
    Windows.Graphics.Display.DisplayProperties.OrientationChanged +=
      wrapper.WeakOrientationChangedHandler;
  }

  void OrientationChangedHandler(object sender)
  {
    // Do some stuff
  }

  class WeakDelegateWrapper : WeakReference<LargeClass>
  {
    DisplayPropertiesEventHandler wrappedHandler;

    public WeakDelegateWrapper(LargeClass wrappedObject,
      DisplayPropertiesEventHandler handler) : base(wrappedObject)
    {
      wrappedHandler = handler;
      wrappedHandler += WeakOrientationChangedHandler;
    }

    public void WeakOrientationChangedHandler(object sender)
    {
      LargeClass wrappedObject = Target;

      // Call the real event handler on LargeClass if it still exists
      // and has not been garbage collected. Remove the event handler
      // if LargeClass has been garbage collected so that the weak
      // delegate no longer leaks
      if(wrappedObject != null)
        wrappedObject.OrientationChangedHandler(sender);
      else
        wrappedHandler -= WeakOrientationChangedHandler;
    }
  }
}
```

Figure 7 **Implementing an Event Handler with a Lambda**

```
<MainPage x:Class="App.MainPage" ...>
  ...
  <TextBlock x:Name="displayTextBlock" ... />
  <Button x:Name="myButton" ... />
  ...
</MainPage>

public sealed partial class MainPage : Page
{
  ...
  protected override void OnNavigatedTo
  {
    myButton.Click += => (source, e)
    {
      DateTime currentTime = DateTime.Now;
      this.displayTextBlock.Text = currentTime.ToString();
    }
  ...
}
```

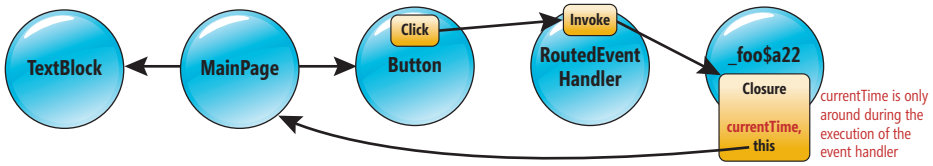


Figure 8 **References Created by the Lambda**

Using a lambda also creates a cycle. The first references are still obviously created from the Page to the Button and the TextBlock (like the top diagram in **Figure 3**).

The next set of references, illustrated in **Figure 8**, is invisibly created by the lambda. The Button's Click event is hooked up to a RoutedEventHandler object whose Invoke method is implemented by a closure on an internal object created by the compiler. The closure must contain references to all variables referenced by the lambda. One of these variables is "this," which—in the context of the lambda—refers to the Page, thus creating the cycle.

If the lambda is written in C# or Visual Basic, the CLR GC will reclaim the resources involved in this cycle. However, in C++/CX this kind of reference is a strong reference and *will* cause a leak. This doesn't mean that all lambdas in C++/CX leak. A circular reference wouldn't have been created if we hadn't referenced "this" and only used variables local to the closure when defining the lambda. As one solution to this problem, if you need to access a variable external to the closure in an inline event handler, implement that event handler as a method instead. This allows the XAML compiler to create a weak reference from the event to the event handler and the memory will be reclaimed. Another option is to use pointer-to-member syntax, which allows you to specify whether a strong or weak reference is taken against the class containing the pointer-to-member method (in this case, the Page).

Use the Event Sender Parameter

As discussed in the previous article, each event handler receives a parameter, typically called "sender," which represents the event source. The event source parameter of a lambda helps avoid circular references. Let's modify our example (using C++/CX) so the button shows the current time when it's clicked (see **Figure 9**).

The updated lambda creates the same circular references illustrated in **Figure 8**. They will cause C++/CX to leak, but this can be avoided by using the source parameter instead of referencing myButton through the "this" variable. When the closure method is executed, it creates the "source" and "e" parameters on the stack. These variables live only for the duration of the method call instead of for as long as the lambda is attached to the Button's event handler (currentTime has the same life span). Here's the code to use the source parameter:

```
MainPage::MainPage()
{
  ...
  myButton->Click += ref new RoutedEventHandler([](Platform::Object^ sender,
  Windows::UI::Xaml::RoutedEventArgs^ e)
  {
    DateTime currentTime ;
    Calendar^ cal = ref new Calendar();
    cal->SetToNow() ;
    Button ^btn = (Button^)sender ;
    btn->Content = cal->SecondAsString();  });
  ...
}
```

The references now look like what's shown in **Figure 10**. The reference depicted in red, creating the cycle, is present only during the execution of the event handler. This reference is destroyed once the event handler has completed and we're left with no cycles that will cause a leak.

Use WRL to Avoid Leaks in Standard C++ Code

You can use standard C++ to create Windows Store apps, in addition to JavaScript, C#, C++/CX and Visual Basic. When doing so, familiar COM techniques are employed, such as reference counting to manage the lifetime of objects and testing HRESULT values to determine whether an operation succeeded or failed. The Windows Runtime C++ Template Library (WRL) simplifies the process of writing this code (bit.ly/P1rZrd). We recommend you use it when implementing standard C++ Windows Store apps to reduce any bugs and memory leaks, which can be extremely difficult to locate and resolve.

> Many people find it easier to implement event handlers with a lambda—or inline function—instead of a method.

Use Event Handlers That Cross Language Boundaries with Caution

Finally, there's one coding pattern that requires special attention. We've discussed the possibility of leaks from circular references that involve event handlers, and that many of these cases can be detected and avoided by platform-supplied mitigations. *These mitigations do not apply when the cycle crosses multiple garbage-collected heaps.*

Let's take a look at how this might happen, as shown in **Figure 11**.

This example is very similar to the previous examples. A Page contains a TextBlock that displays a little information. In this sample, though, the TextBlock displays the user's location, as shown in **Figure 12**.

Figure 9 **Making the Button Show the Current Time**

```
<MainPage x:Class="App.MainPage" ...>
  ...
  <Button x:Name="myButton" ... />
  ...
</MainPage>

MainPage::MainPage()
{
   ...
   myButton->Click += ref new RoutedEventHandler(
     [this](Platform::Object^ sender, Windows::UI::Xaml::RoutedEventArgs^ e)
   {
     Calendar^ cal = ref new Calendar();
     cal->SetToNow() ;
     this->myButton->Content = cal->SecondAsString();
   });
   ...
}
```

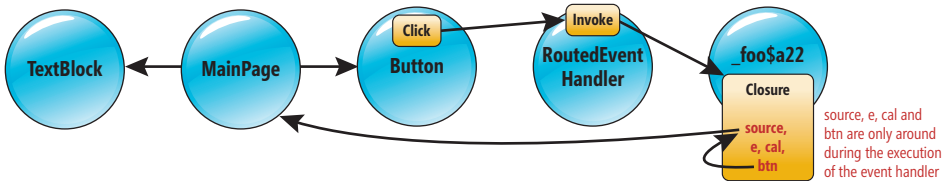


Figure 10 **Using the Source Parameter**

At this point, you probably could've drawn the circular references yourself. What's not obvious, however, is that the circular references span a garbage-collector boundary. Because the references extend outside the CLR, the CLR GC can't detect the presence of a cycle and this will leak. It's difficult to prevent these types of leaks because you can't always tell in which language an object and its events are implemented. If Geolocator is written in C# or Visual Basic, the circular references will stay within the CLR and the cycle will be garbage-collected. If the class is written in C++ (as in this case) or JavaScript, the cycle will cause a leak.

> The first step in fixing a memory leak is to identify steady memory growth from operations that should be memory neutral.

There are a few ways to ensure your app isn't affected by leaks like this. First, you don't need to worry about these leaks if you're writing a pure JavaScript app. The JavaScript GC is often smart enough to track circular references across all WinRT objects. (See the previous article for more details on JavaScript memory management.)

You also don't need to worry if you're registering for events on objects you know are in the XAML framework. This means anything in the Windows.UI.Xaml namespace and includes all of the familiar FrameworkElement, UIElement and Control classes. The CLR GC is smart enough to track circular references through XAML objects.

The other way to deal with this type of leak is to unregister the event handler when it's no longer needed. In this example, you could unregister the event handler in the OnNavigatedFrom event. The reference created by the event handler would be removed and all of the objects would get destroyed. Note that it's not possible to unregister lambdas, so handling an event with a lambda can cause leaks.

Figure 11 **Displaying a User's Location**

```
<Page x:Class="App.MainPage" ...>
  ...
  <TextBlock x:Name="displayTextBlock" ... />
  ...
</Page>


public sealed partial class MyPage : Page
{
  ...
  Geolocator gl;

  protected override void OnNavigatedTo{} ()
  {
    Geolocator gl = new Geolocator();
    gl.PositionChanged += UpdatePosition;
  }

  private void UpdatePosition(object sender, RoutedEventArgs e)
  {
    // Change the text of the TextBlock to reflect the current position
  }
  ...
}
```

Analyzing Memory Leaks in Windows Store Apps Using C# and Visual Basic

If you're writing a Window Store app in C# or Visual Basic, it's useful to note that many of the techniques discussed in the previous article on JavaScript apply to C# and Visual Basic as well. In particular, the use of weak references is a common and effective way to reduce memory growth (see bit.ly/S9gVZW for more information), and the "Dispose" and "Bloat" architecture patterns apply equally well.

Now let's take a look at how you can find and fix common leaks using tools available today: Windows Task Manager and a managed code profiling tool called PerfView, available for download at bit.ly/UTdb4M.

In the "Event Handlers" section of the previous article on leaks, we looked at an example called LeakyApp (repeated in **Figure 13** for your convenience), which causes a memory leak in its window's SizeChanged event handler.

In our experience, this is the most common type of leak in C# and Visual Basic code, but the techniques we'll describe apply just as well to circular event leaks and to unbounded data-structure growth. Let's take a look on how you can find and fix leaks in your own apps using tools available today.

Looking for Memory Growth

The first step in fixing a memory leak is to identify steady memory growth from operations that should be memory neutral. In the "Discovering Memory Leaks" section of the previous article, we discussed a very simple way you can use the built-in Windows Task Manager to watch for the growth of the Total Working Set (TWS) of an app by running through a scenario multiple times. In the example app, the steps to cause a memory leak are to click on a tile and then navigate back to the homepage.

In **Figure 14**, the top screenshot shows the working set in Task Manager before 10 iterations of these steps, and the bottom screenshot shows this after 10 iterations.

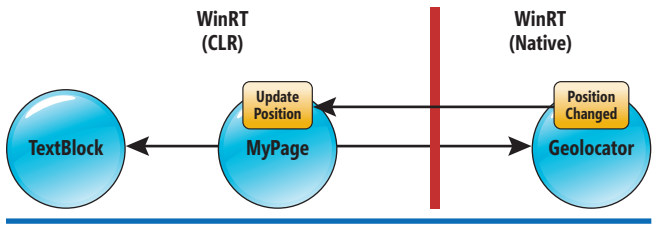


Figure 12 **Circular References Span a Garbage-Collector Boundary**

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

LIVE! 360
IT EVENTS WITH PERSPECTIVE

Orlando, FL December 10-14

Royal Pacific Resort at Universal Orlando | **vslive.com/orlando**

Visual Studio Live! provides education and training on what's now, new and next on the .NET development platform.

Session Topics Include:

- ASP.NET
- HTML5
- Metro
- Mobile
- TFS / ALM
- Visual Studio / .NET
- Windows 8 / WinRT
- Windows Phone
- WPF / Silverlight
- XAML

Last year's event sold out…

Register Today!
Use Promo Code VSNOV
vslive.com/orlando

LIVE! 360
IT EVENTS WITH PERSPECTIVE

Buy 1 Event, Get 3 Free!

Visual Studio Live! is co-located with:

Cloud & Virtualization LIVE! THE FUTURE OF COMPUTING
SharePoint LIVE! TRAINING FOR COLLABORATION
SQL Server LIVE! TRAINING FOR DBAs AND IT PROS

GOLD SPONSORS: axosoft, New Relic, Symantec, veeam

SUPPORTED BY: Microsoft, msdn magazine, Visual Studio MAGAZINE, Redmond MAGAZINE, VIRTUALIZATION REVIEW

MEDIA SPONSOR: THE CODE PROJECT

PRODUCED BY: 1105 MEDIA

Figure 13 **LeakyApp**

```
public sealed partial class ItemDetailPage : LeakyApp.Common.LayoutAwarePage
  {
    public ItemDetailPage()
    {
      this.InitializeComponent();
    }

    protected override void OnNavigatedTo(NavigationEventArgs e)
    {
      base.OnNavigatedTo(e);
      Window.Current.SizeChanged += WindowSizeChanged;
    }
    private void WindowSizeChanged(object sender,
      Windows.UI.Core.WindowSizeChangedEventArgs e)
    {
      // Respond to size change
    }
// Other code
  }
```

After 10 iterations, you can see the amount of memory used has grown from 44,404K to 108,644K. This definitely looks like a memory leak, and we should dig further.

Adding GC Determinism

To be certain we have a memory leak on our hands, we need to confirm that it persists after full garbage collection cleanup. The GC uses a set of heuristics to decide the best time to run and reclaim dead memory, and it usually does a good job. However, at any given time there could be a number of "dead" objects in memory that haven't yet been collected. Deterministically calling the GC allows us to separate growth caused by slow collection and growth caused by true leaks, and it clears the picture when we look to investigate what objects are truly leaking.

The easiest way to do this is to use the "Force GC" button from within PerfView, shown in the next section in the instructions on taking a heap snapshot. Another option is to add a button to your app that will trigger the GCs using code. The following code will induce a garbage collection:

```
private void GCButton_Click(object sender, RoutedEventArgs e)
{
  GC.Collect();
  GC.WaitForPendingFinalizers();
  GC.Collect();
}
```

The WaitForPendingFinalizers and subsequent Collect call ensure that any objects freed up as a result of finalizers will get collected as well.

In the example app, however, clicking this button after 10 iterations freed up only 7MB of the 108MB of the working set. At this point we can be pretty sure there's a memory leak in LeakyApp. Now, we need to look for the cause of the memory leak in our managed code.

Name	Working set (memory)	Memory (shared working set)	Memory (private working set)
LeakyApp.exe	44,404 K	25,168 K	19,236 K

Name	Working set (memory)	Memory (shared working set)	Memory (private working set)
LeakyApp.exe	108,644 K	27,424 K	81,220 K

Figure 14 **Watching for Memory Growth**

Analyzing Memory Growth

Now we'll use PerfView to take a diff of the CLR's GC Heap, and analyze the diff to find the leaked objects.

To find out where memory is being leaked, you'll want to take a snapshot of the heap before and after you run through a leak-causing action in your app. Using PerfView, you can diff the two snapshots to find where the memory growth is.

To take a heap snapshot with PerfView:

1. Open PerfView.
2. Click on Memory in the menu bar.
3. Click Take Heap Snapshot (see **Figure 15**).
4. Select your Windows Store app from the list.
5. Click the "Force GC" button to induce a GC within your application.
6. Set the filename for the dump you wish to save and click Dump GC Heap (see **Figure 16**).

A dump of the managed heap will be saved to the file you specified and PerfView will open a display of the dump file showing a list of all the types on the managed heap. For a memory leak investigation, you should delete the contents of the Fold% and FoldPats text boxes and click the Update button. In the resulting view, the Exc column shows the total size in bytes that type is using on the GC heap and the Exc Ct column shows the number of instances of that type on the GC heap.

Figure 17 shows a view of the GC dump for LeakyApp.

To get a diff of two heap snapshots showing the memory leak:

1. Run through a few iterations of the action that causes the memory leak in your app. This will include any lazy-loaded or one-time initialized objects in your baseline.
2. Take a heap snapshot, including forcing a GC to remove any dead objects. We'll refer to this as the "before" snapshot.
3. Run through several more iterations of your leak-causing action.
4. Take another heap snapshot, including forcing a GC to remove any dead objects. This will be the "after" snapshot.
5. From the view of the after snapshot, click the Diff menu item and select the before snapshot as your baseline. Make sure to have your view opened from the before snapshot, or it won't show up in the diff menu.
6. A new window will be shown containing the diff. Delete the contents of the Fold% and FoldPats text boxes and update the view.

You now have a view that shows the growth in managed objects between the two snapshots of your managed heap. For LeakyApp, we took the before snapshot after three iterations and the after snapshot after 13 iterations, giving the difference in the GC heap after 10 iterations. The heap snapshot diff from PerfView is shown in **Figure 18**.

The Exc column gives the increase in total size of each type on the managed heap. However, the Exc Ct column will show the *sum of the instances in the two heap snapshots* rather than the difference between the two. This is not what you'd expect for this kind of analysis, and future versions of PerfView will allow you to view this column as a difference; for now, just ignore the Exc Ct column when using the diff view.

SharePoint LIVE!

TRAINING FOR COLLABORATION

LIVE! 360
IT EVENTS WITH PERSPECTIVE

Orlando, FL December 10-14

Royal Pacific Resort at Universal Orlando | **splive360.com**

SharePoint Live! is the first event of its kind to share no-hype, practical, independent perspectives about both SharePoint 2010 and SharePoint 2013.

Event Topics Include:

- Strategy, Governance, Adoption
- Management and Administration
- Information & Content Management
- BI, BPA, Search, and Social
- SharePoint and the Cloud
- SharePoint 2010 & SharePoint 2013

Seats are filling up…

Register Today!

Use Promo Code SPNOV

splive360.com

LIVE! 360
IT EVENTS WITH PERSPECTIVE

Buy 1 Event, Get 3 Free!

SharePoint Live! is co-located with:

Cloud & Virtualization LIVE! — THE FUTURE OF COMPUTING

SQL Server LIVE! — TRAINING FOR DBAs AND IT PROS

Visual Studio LIVE! — EXPERT SOLUTIONS FOR .NET DEVELOPERS

GOLD SPONSORS

SUPPORTED BY

MEDIA SPONSOR

PRODUCED BY

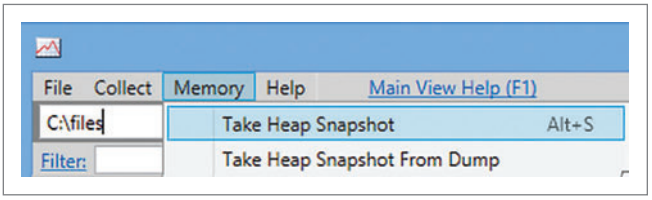


Figure 15 **Taking a Heap Snapshot**

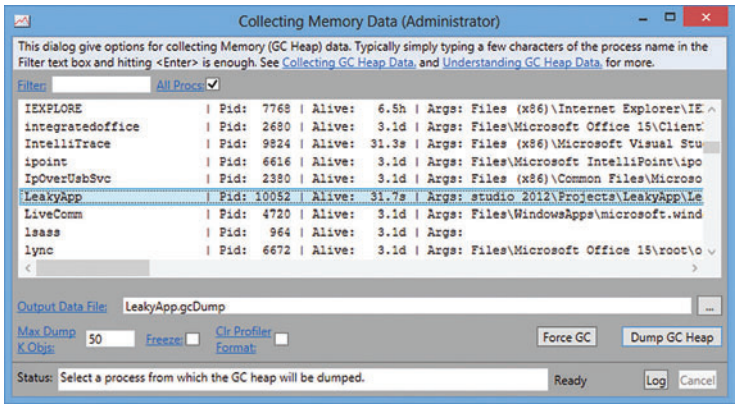


Figure 16 **Dumping the GC Heap**

Any types that leaked between the two snapshots will have a positive value in the Exc column, but determining which object is preventing objects from being collected will take some analysis.

Analyzing the Diff

Based on your knowledge of the app, you should look at the list of objects in the diff and find any types you wouldn't expect to grow over time. Look at types that are defined in your app first, because a leak is likely to be the result of a reference being held on to by your app code. The next place to look is at leaked types in the Windows.UI.Xaml namespace, as these are likely to be held on to by your app code as well. If we look first at types defined only in our app, the ItemDetailPage type shows up near the top of the list. It's the largest leaked object defined in our example app.

Double-clicking on a type in the list will take you to the "Refered-From" (sic) view for that type. This view shows a reference tree of all the types that hold references to that type. You can expand the tree to step through all of the references that are keeping that type alive. In the tree, a value of [CCW (ObjectType)] means that the object is being kept alive by a reference from outside of managed code (such as the XAML framework, C++ or JavaScript code). **Figure 19** shows a screenshot of the reference tree for our suspect ItemDetailPage object.

From this view you can clearly see that the ItemDetailPage is being held live by the event handler for the WindowSizeChanged event, and this is most likely the cause of the memory leak. The event handler is being held on to by something outside of managed code, in this case the XAML framework. If you look at one of the XAML objects, you can see that they're also being kept alive by the same event handler. As an example, the reference tree for the Windows.UI.Xaml.Controls.Button type is shown in **Figure 20**.

From this view, you can see that all of the new instances of UI.Xaml.Controls.Button are being kept alive by ItemDetailPage, which in turn is being kept alive by the WindowSizeChangedEventHandler.

It's pretty clear at this point that to fix the memory leak we need to remove the reference from the SizeChanged event handler to ItemDetailPage, like so:

```
protected override void OnNavigatedFrom(NavigationEventArgs e)
{
  Window.Current.SizeChanged -= WindowSizeChanged;
}
```

After adding this override to the ItemDetailPage class, the ItemDetailPage instances no longer accumulate over time and our leak is fixed.

The methods described here give you some simple ways to analyze memory leaks. Don't be surprised if you find yourself encountering similar situations. It's very common for memory leaks in Windows Store apps to be caused by subscribing to long-lived event sources and failing to unsubscribe from them; fortunately, the chain of leaked objects will clearly identify the problem. This also covers cases of circular references in event handlers across languages, as well as traditional C#/Visual Basic memory leaks caused by unbounded data structures for caching.

In more complex cases, memory leaks can be caused by cycles between objects in apps containing C#, Visual Basic, JavaScript and C++. These cases can be hard to analyze because many objects in the reference tree will show up as external to managed code.

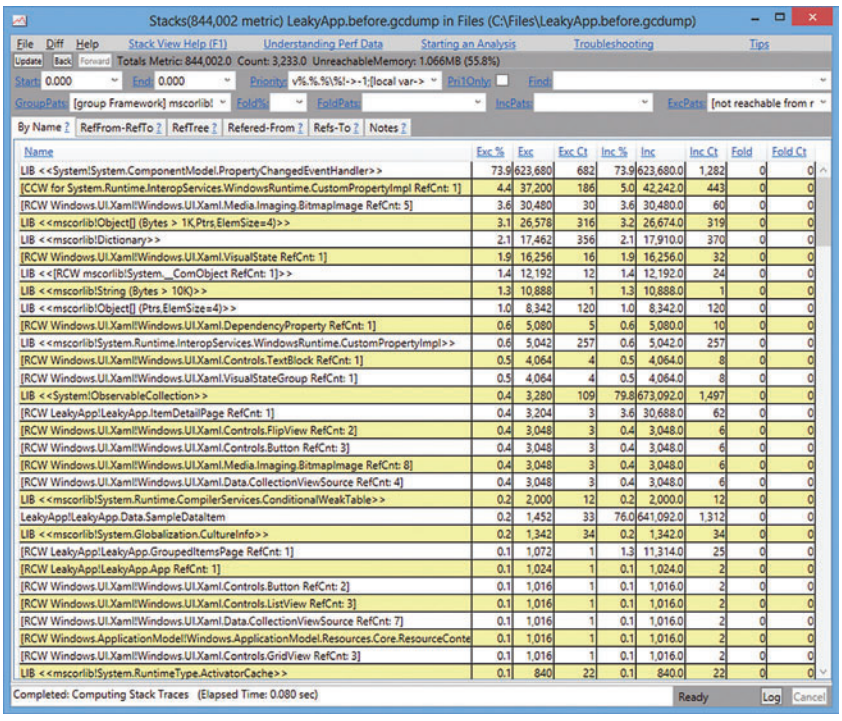


Figure 17 **A Heap Snapshot in PerfView**

SQL Server® LIVE!

TRAINING FOR DBAs AND IT PROS

LIVE! 360
IT EVENTS WITH PERSPECTIVE

Orlando, FL **December 10-14**
Royal Pacific Resort at Universal Orlando | **sqllive360.com**

Education on SQL Server database management, data warehouse/BI model design, Big Data, analytics, performance tuning, troubleshooting and coding against SQL Server.

Event Topics Include:

- BI and Big Data
- Hadoop
- T-SQL, SSIS Packages, Disaster Recovery
- Monitoring, Maintaining, Tuning
- SQL Azure & Windows Azure
- SQL Server 2012

Reserve Your Seat at SQL Server Live!
Register Today!
Use Promo Code SQNOV
sqllive360.com

LIVE! 360
IT EVENTS WITH PERSPECTIVE

Buy 1 Event, Get 3 Free!

SQL Server Live! is co-located with:

Visual Studio LIVE! EXPERT SOLUTIONS FOR .NET DEVELOPERS
Cloud & Virtualization LIVE! THE FUTURE OF COMPUTING
SharePoint LIVE! TRAINING FOR COLLABORATION

GOLD SPONSORS
axosoft New Relic. Symantec. veeam

SUPPORTED BY
Microsoft msdn magazine Visual Studio MAGAZINE Redmond MAGAZINE VIRTUALIZATION REVIEW

MEDIA SPONSOR
THE CODE PROJECT

PRODUCED BY
1105 MEDIA

Considerations for Windows Store Apps That Use Both JavaScript and C# or Visual Basic

For an application that's built in JavaScript and uses C# or Visual Basic to implement underlying components, it's important to remember that there will be two separate GCs managing two separate heaps. This will naturally increase the memory used by the application. However, the most important factor in your app's memory consumption will continue to be your management of large data structures and their lifetimes. Doing so across languages means you need to keep the following in mind:

Measure the Impact of Delayed Cleanup A garbage-collected heap typically contains collectible objects awaiting the next GC. You can use this information to investigate the memory use of your app. If you measure the difference in memory usage before and after a manually induced garbage collection, you can see how much memory was waiting for "delayed cleanup" versus the memory used by "live" objects.

For dual-GC apps, understanding this delta is very important. Due to references between heaps, it might take a sequence of garbage collections to eliminate all collectible objects. To test for this effect and to clear your TWS so that only live objects remain, you should induce repeated, alternating GCs in your test code. You can trigger a GC in response to a button click (for example), or by using a performance analysis tool that supports it. To trigger a GC in JavaScript, use the following code:

```
window.CollectGarbage();
```

For the CLR, use the following:

```
GC.Collect(2, GCCollectionMode.Optimized);
GC.WaitForPendingFinalizers();
```

You might have noticed that for the CLR we use only one GC.Collect call, unlike the code for inducing a GC in the section on diagnosing memory leaks. This is because in this instance we want to simulate the actual GC patterns in your application that will issue only one GC at a time, whereas previously we wanted to try and clean up as many objects as possible. Note that the PerfView Force GC feature shouldn't be used to measure delayed cleanup, because it may force both a JavaScript and a CLR GC.

The same technique should be used to measure your memory use on suspend. In a C#- or JavaScript-only environment, that language's GC will run automatically on suspend. However, in C# or Visual Basic and JavaScript hybrid apps, only the JavaScript GC will run. This might leave some collectible items on the CLR heap that will increase your app's private working set (PWS) while suspended. Depending on how big these items are, your app could be prematurely terminated instead of being suspended (see the "Avoid Holding Large References on Suspend" section of the previous article).

If the impact on your PWS is very large, it may be worth invoking the CLR GC in the suspend handler. However, none of this should be done without measuring a substantial reduction in memory consumption, because in general you want to keep work done on suspend to a minimum (and in particular, nowhere near the 5 second time limit enforced by the system).

Analyze Both Heaps When investigating memory consumption, and after eliminating any impact of delayed cleanup, it's important to analyze both the JavaScript heap and the .NET heap. For the .NET heap, the recommended approach is to use the PerfView tool, described in the "Analyzing Memory Leaks in Windows Store Apps Using C# and Visual Basic" section, whether you want to understand total memory consumption or to investigate a leak.

With the current release of PerfView, you're able to look at a combined view of the JavaScript and .NET heaps, allowing you to see all objects across managed languages and understand any references between them. ■

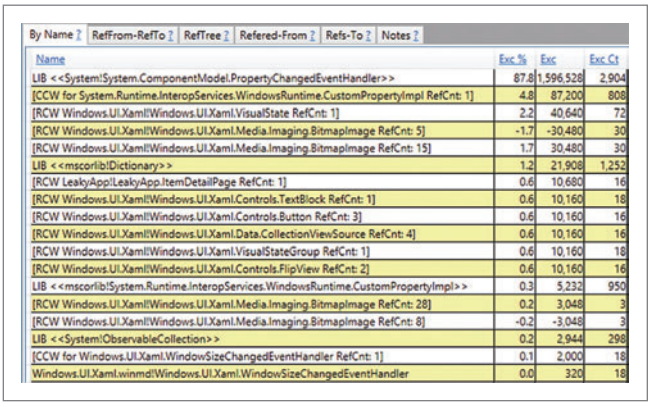


Figure 18 **The Diff of Two Snapshots in PerfView**

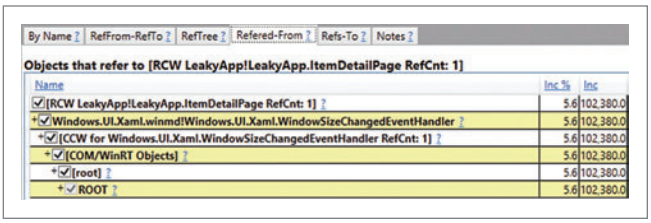


Figure 19 **The Reference Tree for the ItemDetailPage Type in PerfView**

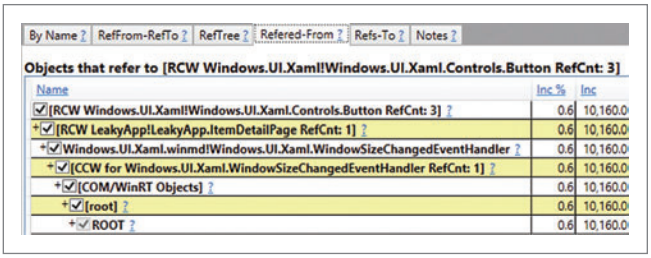


Figure 20 **The Reference Tree for the Windows.UI.Xaml.Controls.Button Type**

Chipalo Street *is a program manager on the Windows 8 XAML team. He started working on Windows Presentation Foundation (WPF) straight out of college. Five years later he's helped XAML evolve through three products (WPF, Silverlight and Windows 8 XAML) and multiple releases. During Windows 8 development, he owned everything related to text, printing and performance for the XAML platform.*

Dan Taylor *is a program manager on the Microsoft .NET Framework team. In the past year, Taylor has been working on performance of the .NET Framework and the CLR for Windows 8 and Core CLR for Windows Phone 8.*

Thanks to the following technical experts for reviewing this article: *Deon Brewis, Mike Hillberg, Dave Hiniker and Ivan Naranjo*

Cloud & Virtualization LIVE!

THE FUTURE OF COMPUTING

LIVE! 360
IT EVENTS WITH PERSPECTIVE

Orlando, FL **December 10-14**

Royal Pacific Resort at Universal Orlando | **virtlive360.com**

Cloud & Virtualization Live! is a new event that provides in-depth training with real-world applicability on today's cloud and virtualization technologies.

Event Topics Include:

- Becoming a Virtualization Expert
- Managing the Modern Cloud Enabled Datacenter
- Tactics in Cloud Application Development
- Constructing & Managing an Application Delivery Infrastructure
- Integrating Mobile Devices and Consumerization with Cloud Computing

Don't Miss Your Chance…

Register Today!

Use Promo Code CVNOV

virtlive360.com

LIVE! 360
IT EVENTS WITH PERSPECTIVE

Buy 1 Event, Get 3 Free!

Cloud & Virtualization Live! is co-located with:

SharePoint LIVE! TRAINING FOR COLLABORATION

SQL Server LIVE! TRAINING FOR DBAs AND IT PROS

Visual Studio LIVE! EXPERT SOLUTIONS FOR .NET DEVELOPERS

GOLD SPONSORS

axosoft · New Relic. · Symantec. · veeam

SUPPORTED BY

Microsoft · msdn magazine · Visual Studio MAGAZINE · Redmond MAGAZINE · VIRTUALIZATION REVIEW

MEDIA SPONSOR

THE CODE PROJECT

PRODUCED BY

1105 MEDIA

Building and Using Controls in Windows Store Apps with JavaScript

Chris Sells and Brandon Satrom

In a previous article, "Data Binding in a Windows Store App with JavaScript," we dug into the Windows Library for JavaScript (WinJS) and its support for data binding. When using data binding, it's often in the context of a control, which is why we spent so much time digging into the care and feeding of the ListView control (you can see that article at msdn.microsoft.com/magazine/jj651576). In this article, we're going to take a quick tour through the controls available to you as a JavaScript programmer building Windows Store apps. And, if those controls don't meet your needs, we'll show you how to build your own.

When using JavaScript to build controls in a Windows Store app, you have access to controls in several families:

- **HTML5 Elements:** HTML5 elements are controls in the sense that they're reusable chunks of UI and behavior, for example, <progress /> and <a />.
- **WinRT Controls:** Controls exposed as Windows Runtime (WinRT) classes projected into JavaScript, such as Windows.UI.Popups.PopupMenu.
- **WinJS Controls:** Controls implemented as JavaScript classes, such as WinJS.UI.ListView.
- **CSS Styles:** CSS provides several styles that allow you to lay out your content items as if they were control containers, for example, column-count: 4.

In this article, we'll focus on the first three categories of controls.

This article discusses:

- HTML5 elements
- WinRT controls
- WinJS controls
- Custom controls

Technologies discussed:

Windows Library for JavaScript, Windows Runtime, HTML5, Windows 8

Code download available at:

archive.msdn.microsoft.com/mag201211Controls

HTML5 Elements

Because Windows Store apps built with JavaScript are based on Web technology, all HTML5 elements work just fine, as **Figure 1** shows.

The details of HTML5 elements are beyond the scope of this article, but we recommend your friendly neighborhood HTML5 documentation for more information. Also, the sample used to create **Figure 1** is provided with the accompanying source code download.

WinRT Controls

The Windows Runtime provides all kinds of functionality in all kinds of areas, but in the case of controls, it provides only two:

- **Message Dialog:** A message with an optional title.
- **Pop-up Menu:** A menu limited to six or fewer items.

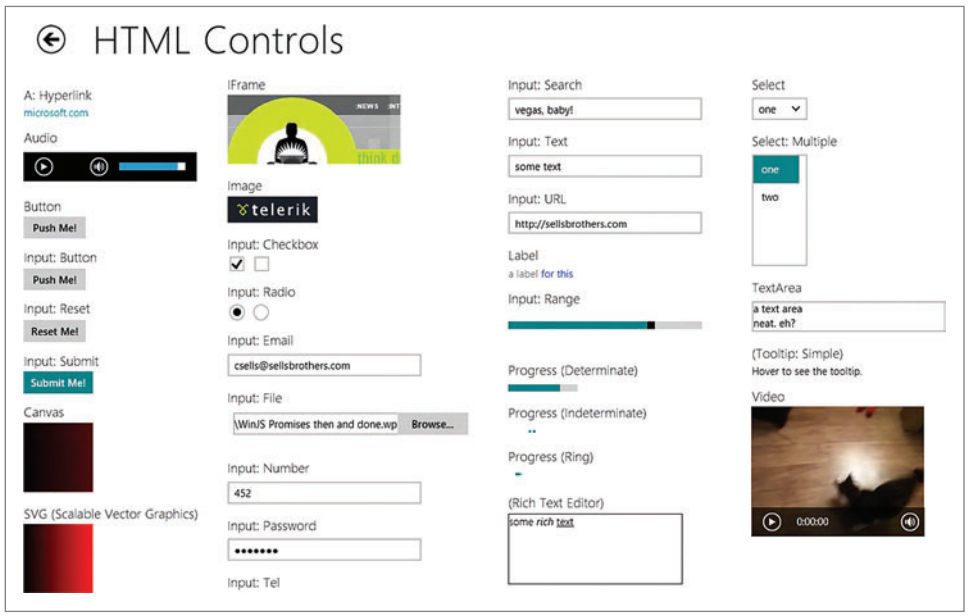


Figure 1 **HTML5 Controls Available to Your Windows Store Apps Built with JavaScript**

The message dialog is invoked using the MessageDialog function:

```
var popups = Windows.UI.Popups;
var mb = new popups.MessageDialog("and welcome to my message box!", "Hello!");
mb.showAsync();
```

The MessageDialog class has a showAsync method that returns a promise, just like all the other async operations a Windows Store app built with JavaScript has at its disposal. In this case, however, we're ignoring the promise because we often don't care when a message dialog goes away. **Figure 2** shows the result (using "MessageBox," which is the previous terminology for MessageDialog).

The PopupMenu class is used similarly:

```
var popups = Windows.UI.Popups;
var menu = new popups.PopupMenu();
menu.commands.push(new popups.UICommand("one", null, 1));
menu.commands.push(new popups.UICommand("two", null, 2));
menu.showAsync({ x: 120, y: 360 }).done(function (e) {
  // Do something with e.label and/or e.id
  ...
});
```

In this example, after creating a PopupMenu object, we provide two UICommand objects, each with a label and optional callbacks and id parameters. We're not using the callback for each of the commands in this example because we're capturing the event parameter in the "done" completion method. A pop-up menu looks like you'd expect, as shown in **Figure 3**.

Remember that as of this writing, the context menu is limited to only six items.

WinJS Controls

Although HTML5 controls are rich and varied, the set won't be extensible until the World Wide Web Consortium decides to add a new element tag and the browser vendors decide to implement it. Likewise, the set of WinRT controls is also not extensible (although you can build non-UI Windows Runtime Components). For the extensible set of controls that have been built specifically with Windows Store apps in mind, the sweet spot is the set provided by WinJS.

A WinJS control is a control implemented in JavaScript that provides a certain signature in the constructor function:

```
function MyControl(element,
  options) {...}
```

The element argument is the HTML Document Object Model (DOM) element that's meant to act as the host for the control's content, often a div. The options argument is a JavaScript object used to provide optional configuration arguments, such as the ListView itemDataSource property.

To see a WinJS control in action, let's consider a div meant to act as the host of a DatePicker control:

```
<div id="datePickerDiv"></div>
```

With this in place, we can create a DatePicker as easily as this:

```
var datePicker = new WinJS.UI.DatePicker(datePickerDiv);
```

And the output is a brand-new DatePicker control, as shown in **Figure 4**.

If we'd like to configure a control, we can pass in a set of options:

```
var datePicker = new WinJS.UI.DatePicker(datePickerDiv, { current: "6/2/1969" });
```

In the case of the DatePicker, the current option lets us set the currently displayed date, as shown in **Figure 5**.

Once you've got a control associated with your HTML5 element, you can grab the control using the winControl property:

```
var datePicker = datePickerDiv.winControl; // Magical well-known property name
datePicker.current = "5/5/1995"; // Now we're talking to the control
```

Further, once you've got the control, you can get back to the associated HTML5 element with the element property:

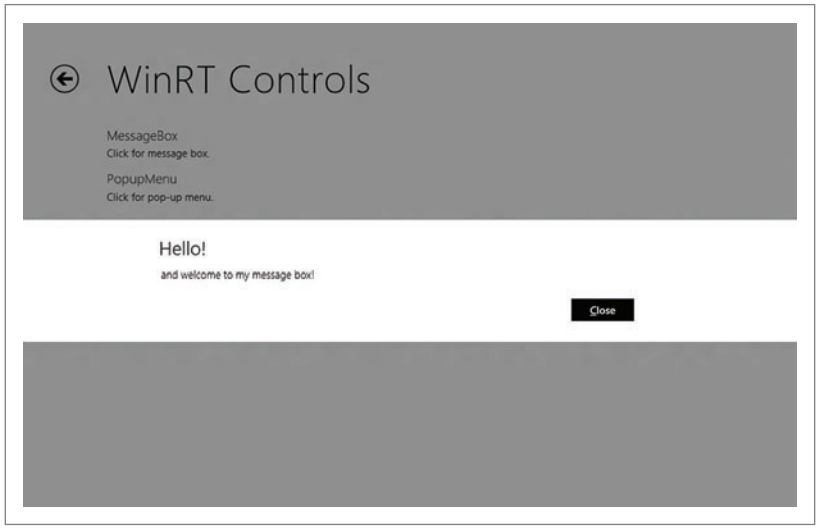


Figure 2 **The WinRT Message Dialog**

```
var datePickerDiv = datePicker.element;
datePickerDiv.style.display = "none";
// Now we're talking to the HTML element
```

In addition to allowing for programmatic creation, each control also provides for declarative creation via the data-win-control and data-win-options attributes, as we've seen:

```
<div id="datePicker2"
  data-win-control="WinJS.UI.DatePicker" data-win-options=
  "{current: '6/2/1969'}" ></div>
```

The data-win-control attribute is the name of the constructor function to call. As in data binding, which requires a call to WinJS.Binding.processAll for the data-win-bind attributes to be parsed (see our previous article), the data-win-control property needs a call to WinJS.UI.processAll for it to be parsed and the controls to be created. This is why you'll see a call to WinJS.UI.processAll in all of the generated project template code.

The data-win-options string is parsed as a less-powerful JavaScript object-initialization syntax. For example, you'll notice that instead of setting the current option for the DatePicker control by creating a Date object, we passed the string directly. That's because the options parser doesn't understand the "new" keyword—it only works with static data. However, because the DatePicker and the other WinJS controls are expecting to be created in a declarative way, they make special allowances for the limitations of the options parser, which—in this case for the DatePicker—means taking in a string and parsing it as a Date object for you.

Every control has a different set of options, and we'll refer you to the documentation to see which controls have which options. **Figure 6** shows a list of the built-in WinJS controls.

Figure 7 shows the WinJS controls in action.

You should feel free to mix and match HTML5 controls, WinRT controls and WinJS controls in your Windows Store app.

Or, if you don't find the control you want on the list provided by HTML5, Windows Runtime or WinJS, you can build your own.

Custom Controls

As we mentioned, a WinJS control is just a function that provides a constructor of the following form:

```
function MyControl(element, options) {...}
```

Building such a control is a matter of implementing a function to create the HTML under the parent element passed in as the first argument and using the options object passed in as the second argument. For example, imagine that we wanted to build a little clock control such as that shown in **Figure 8**.

Assume we have a div all set to contain our clock control:

```
<div id="clockControl1"></div>
```

Just like the built-in WinJS controls, we'd like to be able to create an instance of a custom control, like so:

```
var clock = new Samples.UI.ClockControl(clockControl1, { color: 'red' });
clock.color = 'red'; // Can set options as part of construction or later
```

The name we've picked for our custom control is ClockControl from the Samples.UI namespace. Just as before, creating the control is a matter of passing in the containing element (clockControl1) and an optional set of name/value pairs for options. If later in the lifetime of the control we'd like to change one of the control's options, we should be able to do so by setting an individual property value.

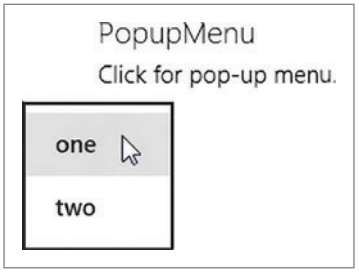


Figure 3 **The WinRT Pop-up Menu**

We'd also like to be able to create custom controls declaratively:

```
<script src="/js/clockControl.js"></script>
...
<div id="clockControl2"
  style="width: 200px; height: 200px;"
  data-win-control="Samples.UI.ClockControl"
  data-win-options="{color: 'red'}">
</div>
```

As part of our implementation, we'd like to make sure that the winControl and element properties are set up, that private members are marked appropriately and that events can be handled appropriately. As we dig into the implementation of the ClockControl, we'll see how WinJS helps us implement these features.

Control Class First, we'll need to make sure that the ClockControl makes it into the right namespace. Most modern languages have the concept of a namespace as a way of separating types, functions and values into separate named areas to avoid collisions. For example, if Microsoft provides a ClockControl type in WinJS 2.0, it'll be in the WinJS.UI namespace, so it won't collide with Samples.UI. In JavaScript, a namespace is just another object with constructors, functions and values, which you could populate like this:

```
// clockControl.js
(function () {

  // The hard way
  window.Samples = window.Samples || {};
  window.Samples.UI = window.Samples.UI || {};
  window.Samples.UI.ClockControl = function(element, options) { ... };

})();
```

This works just fine. However, defining namespaces (and nested namespaces) is such a common thing to do that WinJS (like many JavaScript libraries) provides a shortcut:

```
// clockControl.js
(function () {

  // The easy way
  WinJS.Namespace.define("Samples.UI", {
    ClockControl: function (element, options) { ... };
  };

})();
```

The define function in the WinJS.Namespace namespace allows for defining a new namespace, properly handling the parsing of dotted names for you. The second argument is an object to define the constructors, functions and values that we'd want to expose from this namespace, which is just the ClockControl constructor in our case.

Control Properties and Methods On our ClockControl type, we'd like to expose methods and properties, such as the color

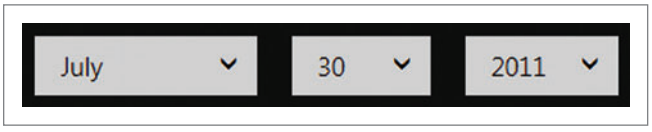


Figure 4 **Output of the DatePicker Control**

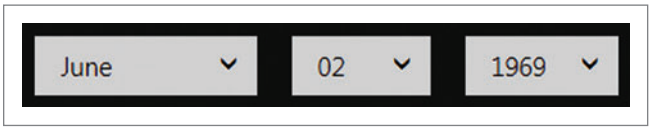


Figure 5 **Setting the Currently Displayed Date**

Figure 6 **The WinJS Controls**

Name	Description	Class
App Bar	Displays app-level commands in a toolbar	WinJS.UI.AppBar
Date Picker	Displays a UI to pick a date	WinJS.UI.DatePicker
Flip View	Flips through a set of content, one item at a time	WinJS.UI.FlipView
Flyout	Displays an overlay with arbitrary content	WinJS.UI.Flyout
List View	Displays a collection of items in a list or grid, grouped or ungrouped	WinJS.UI.ListView
Rating	Displays a UI to rate something, for example, a movie	WinJS.UI.Rating
Semantic Zoom	Provides a UI to zoom from one ListView to another, for example, a grouped ListView zooming out to a list of groups	WinJS.UI.SemanticZoom
Settings Flyout	Provides a UI for configuration of app settings	WinJS.UI.SettingsFlyout
Time Picker	Displays a UI to pick a time	WinJS.UI.TimePicker
Toggle Switch	Displays a UI to pick between two choices	WinJS.UI.ToggleSwitch
Tooltip (Rich)	Displays a tooltip with arbitrary HTML content	WinJS.UI.Tooltip
View Box	Provides a logically fixed-sized region scaled to the available space	WinJS.UI.ViewBox

property. Those methods and properties could be instance or static, and they might be public or private (at least as "private" as JavaScript allows an object's members to get). All of these concepts are supported via the correct use of the constructor's prototype property and the Object.defineProperties method new to JavaScript. WinJS provides a shortcut for this, too, via the define method on the WinJS.Class namespace:

```
WinJS.Namespace.define("Samples.UI", {
  ClockControl: WinJS.Class.define(
    function (element, options) {...}, // ctor

  { // Properties and methods
    color: "black",
    width: { get: function () { ... } },
    height: { get: function () { ... } },
    radius: { get: function () { ... } },

    _tick: function () { ... },
    _drawFace: function () { ... },
    _drawHand: function (radians, thickness, length) { ... },
  })
});
```

In addition to methods and properties, a control often exposes events.

The WinJS.Class.define method takes the function that acts as the constructor, but it also takes the set of properties and methods. The define method knows how to create a property from the get and set functions provided. Further, it knows that properties or methods prefixed with an underscore—for example, _tick—are meant to be "private." JavaScript doesn't really support private methods in the traditional sense—that is, we can still call the _tick method. However, they won't show up in Visual Studio 2012 IntelliSense or in JavaScript for-in loops, which is at least a handy way to signal they aren't meant for public consumption.

The constructor sets up the properties required to be a WinJS control, as shown in **Figure 9**.

The first thing the constructor does is set up the well-known winControl and element properties so a developer can go back and forth between the hosting HTML5 element and the JavaScript control.

Next, the constructor handles the options. You'll recall that the options can be specified as a set of name/value pairs or—using the data-win-options attribute from HTML5—a string. WinJS handles the parsing of the options string into a JavaScript object, so you can just deal with the name/value pairs. If you like, you can pull out individual properties, for example, the color property in our case. However, if you have a large list of options, the setOptions method in the WinJS.UI namespace will traverse all of the properties in the options object and set them as properties on your control. For example, the following blocks of code are equivalent:

```
// Setting each property one at a time
myControl.one = "one";
myControl.two = 2;

// Setting all properties at once
WinJS.UI.setOptions(myControl, {
  one: "one",
  two: 2,
});
```

After setting the control's options, it's the constructor's job to create whatever child elements of the HTML5 parent element it needs to get the job done. In the case of the ClockControl, we're using the HTML5 canvas element and a timer. The implementation of this control is just plain-old HTML and JavaScript, so it isn't shown here (but it is available in the accompanying code download).

Figure 7 **The WinJS Controls in Action**

Figure 8 **A Custom Clock Control**

Control Events In addition to methods and properties, a control often exposes events. An event is some notification from your control that something interesting has happened, such as the user has clicked on the control or the control has reached some state that triggers some other kind of behavior in your program. Based on the example set by the HTML DOM, you're going to want methods such as addEventListener and removeEventListener to allow developers to subscribe to whatever events your control exposes as well as corresponding onmyevent properties.

For example, if we wanted to expose an event from our sample ClockControl every 5 seconds, we'd expect to be able to subscribe to it programmatically:

```
// Do something every 5 seconds
window.clockControl_fiveseconds = function (e) {
  ...
};

var clock = new Samples.UI.ClockControl(...);

// This style works
clock.onfiveseconds = clockControl_fiveseconds;

// This style works, too
clock.addEventListener("fiveseconds", clockControl_fiveseconds);
```

Declaratively, we'd like to be able to attach to custom events, too:

```
<!-- this style works, three -->
<div data-win-control="Samples.UI.ClockControl"
  data-win-options="{color: 'white',
    onfiveseconds: clockControl_fiveseconds}" ...>
</div>
```

Enabling all three of these styles requires two things: the methods to manage event subscriptions (and for dispatching events when they occur) and a property for each event. Both of these are provided by the WinJS.Class namespace:

```
// clockControl.js
...
WinJS.Namespace.define("Samples.UI", {
  ClockControl: WinJS.Class.define(...);
});

// Add event support to ClockControl
WinJS.Class.mix(Samples.UI.ClockControl, WinJS.UI.DOMEventMixin);

WinJS.Class.mix(Samples.UI.ClockControl,
  WinJS.Utilities.createEventProperties("fiveseconds"));
```

Figure 9 **The Constructor Sets up the Properties Required to Be a WinJS Control**

```
WinJS.Namespace.define("Samples.UI", {
  ClockControl: WinJS.Class.define(function (element, options) {
    // Set up well-known properties
    element.winControl = this;
    this.element = element;

    // Parse the options; that is, the color option
    WinJS.UI.setOptions(this, options);

    // Create the drawing surface
    var canvas = document.createElement("canvas");
    element.appendChild(canvas);
    this._ctx = canvas.getContext("2d");

    // Draw the clock now and every second
    setTimeout(this._tick.bind(this), 0);
    setInterval(this._tick.bind(this), 1000);
  },
  ...
});
```

The mix method of WinJS.Class allows you to mix in properties and methods provided by existing objects. In this case, the DOMEventMixin from WinJS.UI provides three methods:

```
// base.js
var DOMEventMixin = {
  addEventListener: function (type, listener, useCapture) {...},
  dispatchEvent: function (type, eventProperties) {...},
  removeEventListener: function (type, listener, useCapture) {...},
};
```

Once we mix in the methods from DOMEventMixin, we can create the properties for each of the custom events using the mix method with an object created by the createEventProperties method of WinJS.Utilities. This method produces the set of events methods for each of the comma-delimited event names you pass in, prepending the "on" prefix. With this set of properties and methods provided by these two calls to the mix method, we've augmented our custom control to support the fiveseconds event. To dispatch an event of this type from inside the control, we call the dispatchEvent method:

```
// clockControl.js
...
_tick: function () {
  var now = new Date();
  var sec = now.getSeconds();
  ...

  // Fire the 5 second event
  if (sec % 5 == 0) {
    this.dispatchEvent("fiveseconds", { when: now });
  }
},
...
```

Calling dispatchEvent takes two parameters: the name of the event and the optional details object available in the event itself. We're passing in a single "when" value, but JavaScript being JavaScript, we could pass whatever we wanted. Accessing the event detail in the handler is a matter of pulling out the detail value of the event object itself:

```
// Do something every 5 seconds
window.clockControl_fiveseconds = function (e) {
  var when = e.detail.when;
  ...
};
```

The principles of defining WinJS controls that we've shown you—defining a class in a namespace, setting the winControl and element properties, processing the options object, defining properties and methods, and defining and dispatching custom events—are the exact same techniques that the WinJS team at Microsoft used to produce the WinJS controls themselves. You can learn a lot about how your favorite controls were built by reading the ui.js file provided with WinJS. ■

Chris Sells *is the vice president of the Developer Tools Division at Telerik. He is coauthor of "Building Windows 8 Apps with JavaScript" (Addison-Wesley Professional, 2012), from which this article was adapted. More information about Sells, and his various projects, is available at sellsbrothers.com.*

Brandon Satrom *is a program manager in the Kendo UI Division at Telerik. He is coauthor of "Building Windows 8 Apps with JavaScript" (Addison-Wesley Professional, 2012), from which this article was adapted. You can follow him on Twitter at twitter.com/BrandonSatrom.*

Thanks to the following technical experts for reviewing this article: *Chris Anderson, Jonathan Antoine, Michael Weinhardt, Shawn Wildermuth and Josh Williams*

Designing Windows Store News Apps

By Nazia Zaman

Windows Store apps provide unparalleled reach across a range of devices, from touch-centered tablets to high-resolution laptop and desktop PCs. These different form factors represent a unique opportunity for news publishers to enable great user experiences across a variety of scenarios and reach a large audience with one app.

This article highlights new capabilities in Windows Store apps that are particularly relevant for news apps, including the following:

Content before chrome: Your content becomes the focus as the OS fades into the background. Users can more readily engage with the content because they aren't distracted by the interface.

Branding opportunities: You can showcase your brand in your app with Live Tiles, a splash screen, your logo and more.

Share contracts and Device contracts: You can use contracts to make your content easier to share and to enable your app to make use of nearby devices.

This article is based on original content hosted on the Windows Dev Center at bit.ly/J8VAkO. Be sure to check this site for the latest content and guidance on this topic.

This article discusses:

- Designing Windows Store news apps
- Typography in Windows Store apps

Technologies discussed:

Windows 8

Notifications: You can use Live Tiles and toast notifications to alert your users to new or updated content. When your content is current and dynamic, users are more likely to engage with your app and use it more frequently.

Layout and Navigation in News Apps

News apps typically present news in many different categories. Using the hierarchical navigation pattern (bit.ly/NMbxOr) for Windows Store apps, you can speed up your app and provide a fluid experience for your users while still featuring your content up front.

Hierarchical Model A news app usually includes a great deal of content the user discovers and consumes. To manage information density, news apps and Web sites also tend to have a lot of different categories. For example, a "breaking news" or "top stories" section is usually featured prominently on the top-level page. Other categories—such as politics, world news, technology, sports, finance and entertainment—likely have some articles displayed at the app's top level. Using the hierarchical model in Windows Store apps for layout and interaction enables your app to bubble interesting content up to the user at the top level, rather than burying it behind tabs or menus. That top-level landing page, or hub, can offer visual variety, engage users and draw users into additional parts of your app.

When you think about the layout of your app, prioritize scenarios and categories based on their relative importance and on how broadly they appeal to the greatest number of users. For example, top news stories or breaking news categories are typically important

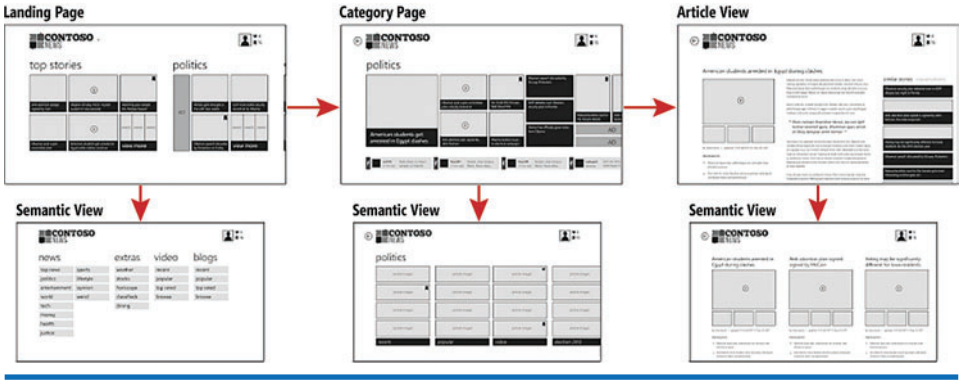


Figure 1 **Hierarchical Model for a News App**

to most users of news apps, so they should be featured prominently in your app's hub page.

Figure 1 illustrates a hierarchical model for a sample news app.

Your App's Hub Page Your app's hub can show top stories, breaking news, content recommended for the user and featured articles for all the different categories in one surface that's easily panned. Each category group can bubble engaging content up to the hub. Tapping a group's header enables the user to drill into a particular section and see more content.

News Categories Lay out all your app categories in the app's hub page and showcase a few articles in each category to draw in the user. You can indicate that a category includes more articles by displaying a count such as "10 more" next to the group header. When the user taps the count, additional sections appear.

Breaking News Breaking news is a key scenario for news apps. You can highlight breaking news and developing stories by featuring them at the front of your hub page, making them bigger than other content on your hub and using type and color variations based on the priority of the news. For example, you could differentiate live events by content placement and size—but also by including a "Live" label in a bright color such as red in the headline or within a picture to draw the user's eye to that content.

Bookmarks/Save for Later Users often save and bookmark articles they want to read later. If your app supports this option in a fast and fluid way, users will continue to use your app because they can seamlessly return to articles they've saved. You can easily add a Bookmarks or Favorites section in your hub page to store all a user's saved articles. **Figure 2** shows an example. Users can easily add articles to the "bookmarked" section and remove them at their discretion.

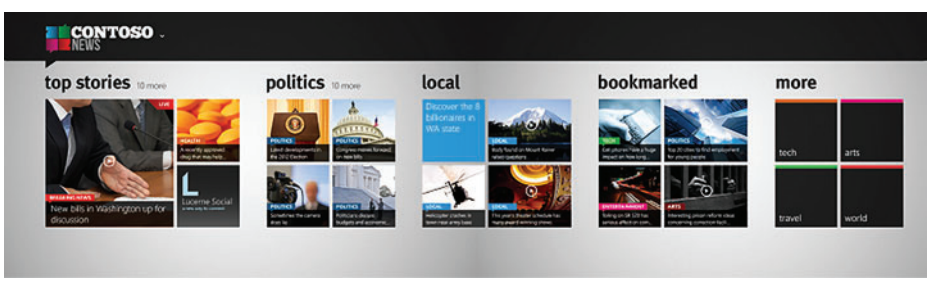


Figure 2 **Bookmarked Section in a Hub**

Dropdown Header Offering a dropdown header enables users to move quickly among the categories in your app. For example, consider a user who's reading a sports article and wants to go to the entertainment section. The user can easily tap a down arrow in the top app bar and see the resulting header menu, as shown in **Figure 3**.

Semantic Zoom Semantic Zoom enables a user to navigate quickly between your app's different news categories within a single view. For example, the user can quickly pinch and pan to get from a top news story on the hub to the last news category on the hub page. The user can also rearrange categories on the hub page by selecting a group and moving it around. This allows him to personalize the app's hub and tailor it to his preferences. **Figure 4** shows how a user can shift the order of categories on the hub page.

Semantic Zoom can also be used on a specific category page to let the user jump quickly from article to article.

> Highlight breaking news and developing stories by featuring them at the front of your hub page.

Article View Your app's article view is where your content is displayed. Your users will spend the majority of their time in your app within this view, so you need to make sure they enjoy the experience. The content must be displayed intuitively, the view must be visually pleasing and the navigation must be efficient.

The article view for a news app should pan horizontally so that users can use their thumbs to move through content easily and ergonomically. Also, consider breaking articles into pages. Page breaks within an article give users a sense of how far they've read. **Figure 5** shows a sample article view for a news app.

As you lay out your news app's article view, consider the different devices your users will view it on, such as a tablet, laptop or desktop PC. Each device benefits from a layout that takes into account the usage patterns of users. For example, when users read news on a tablet, they use both portrait and landscape views. On a laptop or desktop PC, users most likely read in landscape view. When you design your article views, you need to keep both orientations in mind.

Figure 3 **Header Menu**

Figure 4 **Semantic Zoom Enables a User to Reorder Categories on the Hub Page**

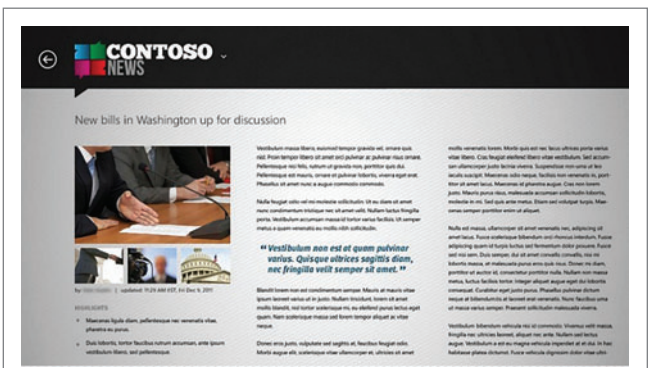


Figure 5 **Article View in a News App**

Figure 6 **Article View Showing Related Articles and Comments**

Reading orientation isn't the only issue. A layout designed for a tablet should be optimized for touch navigation. A touch-centered way to add pages is to use snap points within your article to create "speed bumps" that tell users where they are in the article as they pan through it.

To lay out your article in a visually pleasing way, you should think about using content insets, images or other media elements that span multiple columns rather than having just a set of columns of the same width. You can also add quotes or design tiles to span the entire page to add variety to the layout. Keep in mind that both laptop and desktop users navigate mostly with the keyboard and mouse. You need to make sure your layout responds to both keyboard and mouse input as well as touch.

Using the grid system and the type ramp in news and weather apps is a good idea. They allow you to create a visual hierarchy that enables users to scan and consume a lot of information quickly and easily.

You can display related articles and comments in the article view to get users to continue their reading experience and stay in your app longer, as shown in **Figure 6**.

> Make sure your layout responds to both keyboard and mouse input as well as touch.

Navigating Among Articles Users are often interested in a particular topic or news category and consume multiple articles related to it. If the content in your app tends to be short, let the user pan to the right after the end of the article to get to the next article and pan to the left before the start of the article to get to the previous article. As mentioned earlier, you can also provide Semantic Zoom in this view to enable users to switch to different articles quickly.

If the content of each post or article tends to be long, consider putting Previous Article and Next Article buttons in the top app bar so that the user can easily switch to other articles. You can use these buttons in addition to letting users swipe back and forth between articles. You can play around with different visual styles for such buttons. For example, you could show article thumbnails of the previous and next articles to give users a preview of the content they would be switching to. Alternatively, you could use the standard Previous and Next glyphs but make the top app bar translucent or transparent so that it doesn't distract from the content, as shown in **Figure 7**.

Commanding

Commands for common tasks such as Add to Bookmarks, Add New Feeds and Pin Tile to Start should all live in the bottom app bar to consolidate those commands in one place, making it easier for users to find them reliably. To learn more about commanding, see bit.ly/P35mDj.

Bookmarks Users typically bookmark articles from the hub page when they're saving them to read later, or when they start reading an article but stop before they're finished yet want to come

Figure 7 **Previous and Next Buttons in Top App Bar**

Figure 8 **Example of Font Size, Color and Weight Used to Create the Information Hierarchy**

Figure 9 **Last-Updated Information**

Figure 10 **Hub Page Content Obfuscated During an Update**

back to it. A compelling news app should support both situations, enabling users to bookmark selected articles by using the bottom app bar from both the hub and category pages while reading. The bookmark button should be a toggle—a single button that either adds or removes a bookmark.

Always roam bookmarks so that users can continue to read their bookmarked articles regardless of which device they use. Ensure that the button remains on the left side of the app bar for consistency (even though the button's function is contextual on the hub page but global in the article view).

See bit.ly/NMbxOr for more info about app navigation hierarchy.

Text Selection Enable content selection in your app, especially for text in the article view, because users tend to select excerpts from articles. The system automatically shows the context menu for copying the selected content.

Typography in News Apps

The recommended font for immersive reading scenarios in Windows Store apps is Cambria, which is a serif font that has a more traditional look. You can choose a different font to showcase the personality of your news app as long as the font works with your content and you follow the Windows Store type-ramp guidance regarding size, weight and color. The Segoe UI font recommended for UI elements is appropriate for news and weather content, but you could also use Calibri, a modern sans serif font. The Georgia font is also a popular choice for news Web sites. If you decide to specify alternate system fonts, be sure to confirm that they're installed with Windows 8 and don't require the installation of a separate application such as Microsoft Office. If you use your own custom or licensed fonts, make sure you have sufficient legal rights to include them with your app. Regardless of the font you use, the Windows Store type ramp provides good guidance about the maximum number of sizes and styles you should use.

The type-ramp guidance suggests using a small set of font sizes throughout your app to create a sense of structure and rhythm in your content. If multiple elements use the same font size to convey different types of information—for example, the user name and the reader comments or the article body text and the related-stories header—you need to find some other way, such as through color, font weight and font style, to establish an information hierarchy. **Figure 8** shows how font size, color and weight are used to create the information hierarchy in the sample news app.

See the "Guidelines for Fonts" (bit.ly/NN8Jx2) for more information about choosing fonts in Windows Store apps.

Freshness of Content

In news scenarios, keeping your app's content up-to-date is essential. Several elements indicate fresh content to your users.

Time Content Was Last Updated Displaying "last updated" information to users builds confidence in the reliability of your news app. You should show last-updated information unobtrusively, as illustrated in **Figure 9**. In this example, update information is displayed in the tertiary info style (9-point type) of the type ramp.

Article Posting Time Showing the time an article was posted is another common way to keep users informed of the freshness of

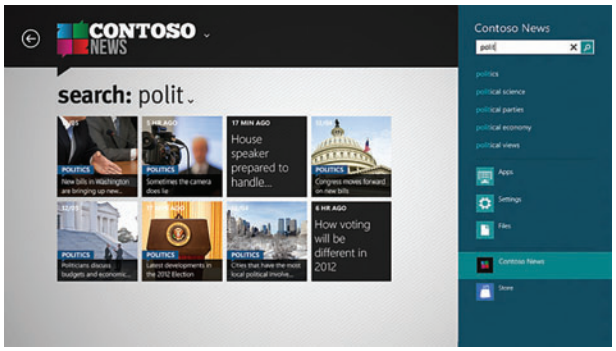


Figure 11 **Sample Search Results**

Figure 12 **Settings Charm**

your app's content. For such time stamps, use the tertiary info type-ramp style as well as color and weight to differentiate the time stamps from other content in your layout that uses the same type ramp.

Refreshing Content To make sure the content in your news app is up-to-date, download data opportunistically. For example, while the user is reading an article, download new content for the hub page so that when the user returns to that page it's up-to-date. Don't download data in the background when the user is using a wireless mobile connection, however, because apps should minimize background data usage when roaming or using mobile broadband. Also, make sure your app can handle offline scenarios by notifying users when content isn't available offline. See "Maintaining Network Connectivity" (bit.ly/PzcZVW) to learn best practices for offline and metered scenarios.

If your app needs to update content for a page that a user is currently on, indicate progress at the top of the canvas area that's being updated, as described in the "Guidelines and Checklist for Progress Controls" (bit.ly/OrT4Tv). Also, use animations to insert new content and remove older content from the canvas, as advised in the "Guidelines and Checklist for List Animations" (bit.ly/SwFGzF).

If your news app content hasn't been updated in a while and the user switches to it, consider obfuscating the old content and showing progress on the canvas to indicate that content is being updated, as shown in **Figure 10**. Make sure the app is responsive even while the update is underway. Also, if the connection is weak, be sure to set a timeout value and display your app's offline UI.

Contracts and Charms

Your news app should use Windows Store contracts whenever possible to help connect your app with the rest of the user's Windows 8 experience. For example, the Share contract provides the user with a familiar and natural way to share content between two apps. You can share content from your app in several ways that can differentiate your news app from others, as described in the following sections.

To learn more about contracts, see "App Contracts and Extensions" (bit.ly/PtPDxy).

Sharing Articles Sharing articles is a key feature of Web news apps. By using the Windows Store Share contract, you enable your users to share news articles with any apps that support this format. This capability makes many compelling scenarios possible for your users, such as sharing an article on social networks, sending articles to friends via e-mail, saving links to notebooks for research and planning, and sharing a link with another device through tap and send. Better yet, with the Share contract, you can enable all these scenarios without having to code third-party integration into your app.

To learn more about using the Share contract, see "Adding Share" (bit.ly/NuJWmS).

Comments and Integration with Other Social Apps You can add to the appeal of your news apps by enabling users to post comments or discuss articles and tag them using apps that are integrated with the Share charm. By relying on apps that integrate with the Share charm, you tailor your app's sharing experience to the apps your users actually use for sharing. You also consolidate the user's sharing experience. You don't need to spend time coding for specific social networks because when new social networks are launched your app is integrated with them automatically.

Search Search is another important activity in news apps, which usually display articles from multiple categories. The Windows Store Search charm provides a centralized search experience for users while helping them make sense of different categories of results, as shown in **Figure 11**.

If your app has advanced search scenarios, with filtering and scoped searches, feature these options on the app canvas as appropriate. Also, provide automatic query suggestions as users are entering the search term to make it faster for them to search without typing an entire string.

To learn more about search, see "Guidelines and Checklist for Search" (bit.ly/OrVLo0).

Settings All applicable settings for news apps—such as user-account info, subscription info, privacy settings and notification settings—should be in the Settings charm. The Settings charm provides a single place for users to customize their settings and prevents your app's UI from becoming cluttered. **Figure 12** shows a sample Settings charm.

See "Adding App Settings" (bit.ly/R6Yrwf) for information on how to implement your Settings contract so that users can access your app's settings from the Settings charm.

Devices Devices can be an interesting contract for your news app to plug into. If your app associates a lot of media (such as videos) with news articles, your users might want to view that

Figure 13 **Devices Charm**

media on televisions in shared-media experiences. If your users typically print news articles, you can provide a seamless printing experience for them via the Devices charm, as shown in **Figure 13**.

Connected and Alive

Live Tiles and notifications enable fresh content to display when users are on your Start screen. These features can help you build a connection with your users and make your app alive with activity.

Tiles and Notifications Showing the latest news stories and breaking news on tiles visually draws users into your app. When users find interesting news stories on an app tile, they're likely to tap the tile. Another way of drawing users to your app—and ensuring that they return—is to enable them to pin different news category tiles to Start so they can see notifications and quickly access the categories and articles in which they're interested.

You can also allow users to receive toast notifications of breaking news, which will be displayed even when your app isn't running. Users can enable these notifications through the Settings charm. Make sure to give your users the choice to opt into these notifications. Unsolicited notifications might annoy users who didn't sign up to see them.

Roaming Enabling roaming in your news app provides users who have more than one Windows PC a consistent experience across all their devices. You can roam app settings, article bookmarks, favorite news categories, reading preferences and any other data you think your users want. Consult the "Guidelines for Roaming Application Data" (bit.ly/PzhS1n) to learn more about best practices for roaming app data.

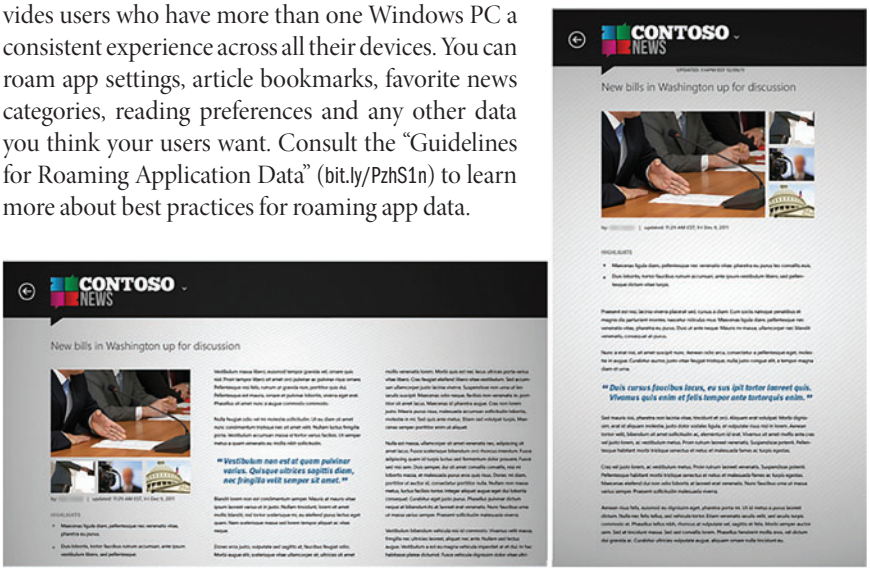


Figure 14 **News App in Landscape and Portrait View**

Orientation and Views

As you design your news app for Windows Store, consider all of the views for your app, such as various screen resolutions and device sizes.

Portrait and Landscape Views The new Windows UI makes it easy to scale your design to both portrait layout and high-resolution screens by including more content in your app for larger devices. **Figure 14** shows the sample news app in both landscape and portrait view. In portrait view, your app should continue to pan horizontally. Portrait view is a common orientation for reading and consuming large amounts of content. You should provide a portrait view for your entire app, but especially for your article view.

Find out more about how your app should support panning at "Touch Interaction Design" (bit.ly/QJAXJv). Go to "Designing Flexible Layouts" (bit.ly/Q7WevF) for more info about view states, user interactions and screen orientations.

Snapped View Windows Store apps let users multitask by "snapping" an app beside another app. Snapped view is a great way to increase your app's time on screen and engage users for longer periods. Having a snapped state for each view of your app enables users to multitask while using your app—that is, your app is on their screen even when they're doing other things.

Consider updating the content in the snapped view by using a FlipView control, which automatically moves through content items and helps the user monitor multiple items at once.

Splash Screen As mentioned earlier, news apps must frequently download fresh Web content to stay current. To make your app fast and responsive, don't wait until all the images for each category have been downloaded before you show the landing page. Waiting can cause users to become impatient. Show users the app's hub page as soon as the article titles have been downloaded, and then asynchronously load images and other data. During the short wait for the hub page to appear, display your splash screen along with a message that informs users that your app is downloading content.

Selling Your App

This article has taken you on a brief tour of the elements that make up a successful Windows Store news app. By following the suggestions provided, you can start creating your own vision of a news app using the latest and greatest technology. The new Windows Store for Developers (bit.ly/vgkLO3) provides opportunities for you to distribute, promote and make money on your apps while providing flexibility if you want to run your own commerce engine for subscriptions. ■

Nazia Zaman *is a program manager on the Core User Experience team for Windows. She graduated from Stanford University with a master's in computer science. Zaman is passionate about human computer interaction, UX, design and astronomy.*

Thanks to the following technical experts for reviewing this article: *The Microsoft User Experience Team for Windows*

Implementing Build Automation with Team Foundation Service Preview

Mario Contreras, Tim Omta and Tim Star

In this article you'll find out how to implement build automation with Team Foundation Service Preview (TFS Preview). As Visual Studio ALM Rangers, we've been using TFS Preview to run builds entirely in the cloud since March. Although support is in place for all of the Team Build features—including gated check-ins and custom build definitions—our focus in this article is not on the basic build capabilities of TFS Preview. Rather, we'll show how to set up build automation on-premises for scenarios where the development team needs more control over the configuration of the build machine than is available on Windows Azure.

For detailed information about how to execute a build on TFS Preview, see Aaron Bjork's "Visual Studio ALM + Team Foundation Server Blog" at bit.ly/H9epDn.

This article discusses a preview edition of Team Foundation Service. All related information is subject to change.

This article discusses:

- Connecting on-premises or to a host
- On-premises configuration
- Configuring a firewall
- Configuring Team Build definitions

Technologies discussed:

Team Foundation Service, Visual Studio, Windows Azure

To recap, the Rangers are a group of experts who promote collaboration between the Visual Studio product group, Microsoft Services and the Microsoft Most Valuable Professional (MVP) community by addressing missing functionality, removing adoption blockers, and publishing best practices and guidance based on real-world experience.

Connecting TFS Preview to Team Foundation Build Service

To automate builds of your software projects, you use Team Foundation Build Service to create a build machine that runs either on-premises or in a hosted scenario. Once your build machine is set up, you need to create a build definition in Visual Studio with instructions about which code projects to compile and how to configure many other options. You must first choose your deployment scenario for Team Foundation Build Service and set up your build machine appropriately.

Preparing for the Installation Before you can install Team Foundation Build Service on a machine connected to TFS Preview, you need to perform the following tasks:

- Make sure to use the Team Foundation Build Service available with Team Foundation Server 2012. The Team Foundation Server 2010 build agent and controller are not compatible with Team Foundation Service on Windows Azure Preview. You must use the build server software that's part of Team Foundation Server 2012.
- Subscribe to TFS Preview.

- Identify the URL to your Project Collection on TFS Preview.
- Configure your user credentials to have "Edit collection-level information" permissions, which are included in the default permissions for the Project Collection Build Administrators groups.
- Configure your firewall to allow inbound connections on TCP port 9191 on the hosted virtual machine (VM) firewall so TFS Preview can communicate with the build service. To ensure the security and integrity of your build machine, always keep Windows Firewall turned on.

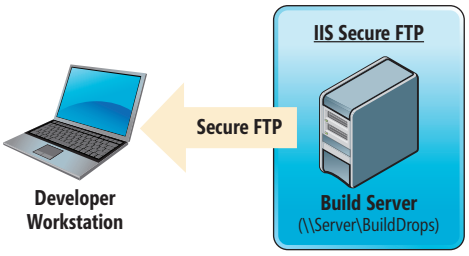


Figure 1 **Secured Build Drop Access**

Installing, Configuring and Executing The process of deploying Team Foundation Build Service on a hosted physical machine or a VM tied to TFS Preview is almost identical to that of deploying an on-premises Team Foundation Server, as described in the MSDN Library article, "How to: Install Team Foundation Build Service," at bit.ly/Oazdwd and the ALM Rangers "Installing Build for Team Foundation Service" guide at go.microsoft.com/fwlink/?LinkID=251576.

To deploy an on-premises build server that leverages TFS Preview in the cloud, we need to borrow some concepts and industry practices from Web hosting.

The main difference is how you secure the build drop. Although you can create a build drop shared folder on the hosted physical machine or the VM, you shouldn't expose the shared folder directly to the Internet. You don't want to increase the attack surface of the build server. As illustrated in **Figure 1**, using the Secure FTP feature of IIS and publishing the build drop folder is probably the best method for making your builds securely available over the Internet.

Dealing with Performance, Bandwidth and Download Concerns Build automation is generally an I/O-intensive operation. When you consider that most builds will download the complete source code of a solution, available bandwidth and download caps are an important factor. To reduce the number of files downloaded during each build, you should consider setting up Team Foundation Server Proxy. The proxy enhances network throughput by caching copies of source control files in a remote location. By caching copies on-premises, the proxy helps the build process avoid a costly download of the files. Caching copies on-premises definitely reduces bandwidth consumption, which will affect the download caps imposed by your Internet Service Provider, as well as improve the overall performance of the build process.

Configuring On-Premises Build Servers

To deploy an on-premises build server that leverages TFS Preview in the cloud, we need to borrow some concepts and industry practices from Web hosting. In most corporate Web hosting environments, a perimeter network (also known as a demilitarized zone—or DMZ—or as a screened subnet) is used to create a secured network or enclave. The perimeter network separates servers that are exposed to the Internet from the corporate network.

A perimeter network can be implemented in a couple of ways. Traditionally, it's implemented by using two physically separate firewalls, each restricting the traffic flow to and from the networks, as illustrated in **Figure 2**.

A more modern approach to implementing a perimeter network uses a secured Virtual Local Area Network (VLAN) and relies on a single firewall that controls all network traffic, as shown in **Figure 3**.

Firewall Configuration for TFS Preview and Team Foundation Build Service Communication

Because of the numerous variations in firewall configuration among the various manufacturers, we're going to limit our discussion to the conceptual level. In this context, two aspects of firewall configuration are important: publishing and access rules.

Publishing refers to the act of opening TCP and User Datagram Protocol (UDP) ports on the external-facing interface or wide area network of the firewall to make services such as Web access, e-mail and Virtual Private Networking available on the Internet. To maintain a secured build server, you need to define publishing in your firewall in the following way:

- Interface: External
- Port Number: 9191
- TCP/UDP: TCP
- Scope: All networks (Internet)

Notice that the scope is set to "All networks (Internet)." You might want to consider limiting the scope of the publishing to the IP set of TFS Preview to prevent other clients or hosts on the Internet from accessing this machine. Adding traffic restrictions adds a second layer of security. You can obtain the latest IPs of TFS

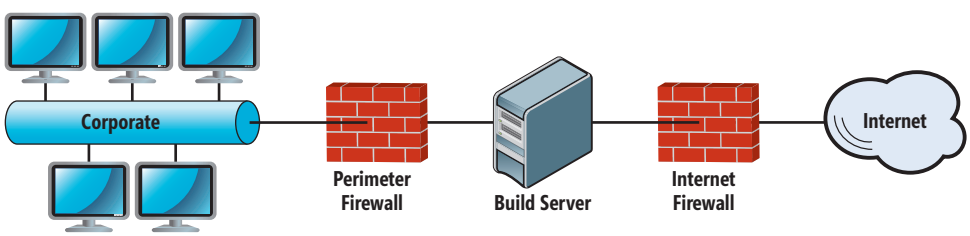


Figure 2 **A Perimeter Network with Two Separate Firewalls**

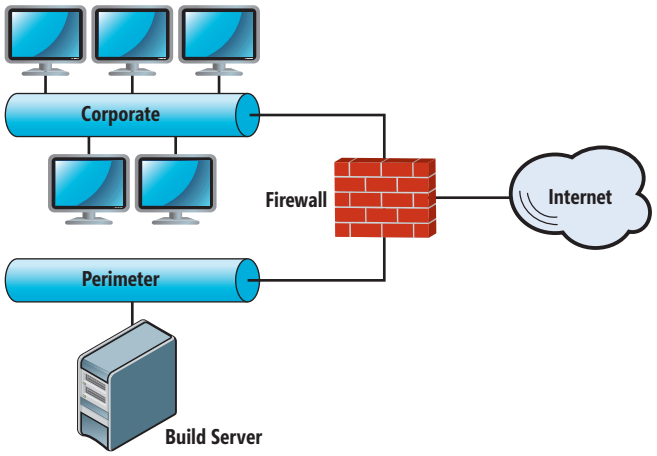


Figure 3 **A Perimeter Network with Virtual Local Area Networks**

Preview by doing a DNS name server lookup. Keep in mind that IPs can change at any time. Some firewalls have the ability to do a reverse lookup against a host's fully qualified domain name to help you monitor current IPs.

Access rules are the constraints imposed on network traffic between the internal and external interfaces of a firewall. To maintain a secured build server, you need to define the access rules in your firewall as shown in **Figure 4**.

> With TFS Preview, you don't have complete control over the build machine.

Detecting Intrusions As with any Web hosting infrastructure, you need to consider implementing intrusion detection when you publish a Team Foundation Build Service server to the Internet.

Securing TFS Preview and Team Foundation Build Service Communication You can strengthen the security of your on-premises build server deployment connected to TFS Preview by configuring the services it uses. We recommend that you configure your deployment to require HTTPS with SSL to maximize the security of your deployment.

To configure the services to use HTTPS with SSL, you need the following:

- An SSL certificate issued by a commercial certificate authority (CA).
- A registered Internet domain name.
- A DNS host name record or alias (CNAME) resolving to the external IP (Internet-facing) of the build server using your registered domain name. (To procure an SSL certificate from a commercial CA, you need a registered domain name.)
- Local Administrator privileges on the build server.
- An internal IP address mapped to the external IP address for the build server on your firewall.

After you've met these requirements, you need to open TCP Port 9191 on your firewall and forward it to the internal IP address of your build server and then configure Team Foundation Build Service to use SSL.

Instructions on how to configure SSL for Team Foundation Server 2010 Build Service are available in the MSDN Library subtopic, "Installing the Certificate on Build Servers," at bit.ly/OA10Vz.

Setting Up Authentication To run the Build Service Configuration Wizard and add a Team Foundation Build Service instance to TFS Preview, you need two user accounts:

- Windows Live ID mapped to a Project Collection on a TFS Preview account (subscription)
- Local or domain user account to use as a service account on the build server

During the initial configuration phase, the Build Service Configuration Wizard uses the Windows Live ID to gain access to TFS Preview and establish a mapping between the Project Collection and the build server. Subsequently, you need to provide the service account credentials for either a domain or a local user account. Using a domain user account as a service account is the best security option. Domain user accounts offer a simpler approach when using a folder share on another server.

Monitoring Builds You can monitor your builds through the Build Notifications power tool that's part of the Team Foundation Server 2010 Power Tools (December 2011) release. **Figure 5** shows the Build Notifications Options dialog box. If you're working with Visual Studio 2012, you need to install Team Explorer 2010 SP1 and the GDR compatibility update before you install Team Foundation Server 2010 Power Tools. You don't need to worry about version conflicts because side-by-side installation of Visual Studio 2012 and Visual Studio 2010 is supported.

Configuring Team Build Definitions with the Hosted Build Controller

TFS Preview provides a pool of build machines that can compile, test and package your application.

Configuring a Hosted Build Controller To use a hosted build controller, the only configuration necessary is that your build definition must specify the hosted build controller in the Build Defaults tab, as shown in **Figure 6**.

Dealing with Constraints With TFS Preview, you don't have complete control over the build machine, which means that if you have an external dependency in your project, you need to either check it in or enable it to be downloaded from a public NuGet feed.

The Visual Studio 2012 and Visual Studio 2010 SP1 (which requires the GDR compatibility update) development environments are installed on the TFS Preview build image, meaning that only projects created with these versions of Visual Studio are supported.

Figure 4 **Defining Access Rules**

Source	Destination	Protocol	Port	Description
Internal Network (Build Server IP)	External Interface (Internet)	HTTPS	8080	Secured communications with TFS Preview
External Interface (Internet)	Build Server (IP)	HTTPS	9191	Team Foundation Build Service endpoint

If you want to build Windows Store apps, you need to install the build service on Windows 8. For more information on using Team Foundation Build to build and test a Windows Store app, go to bit.ly/T3tFSt.

Unit Testing TFS Preview supports unit testing as a part of the build. Unit tests using MSTest work without additional configuration. To execute tests using other unit-testing frameworks, such as NUnit or xUnit, you need to verify that the corresponding test adapter assemblies are checked into source control and then set the build controller's "Version Control Path to Custom Assemblies" property to this location.

Identifying Complex Build Scenarios Deciding which build configuration is best for you depends on the complexity of your build. For straightforward build scenarios with no automated deployment, the build service hosted in TFS Preview is sufficient. For build processes that are more complex, you should use an on-premises build configuration. A complex build configuration would include any build template that meets at least one of these criteria:

- Has automated deployment activities in your workflow
- Contains solutions that were created in versions of Visual Studio earlier than Visual Studio 2010 SP1
- Calls Visual Studio Lab Management
- Interacts with an on-premises machine
- Requires a testing framework without a Visual Studio test adapter

Solutions that require third-party assemblies can be built on TFS Preview as long as you check those assemblies into version control or enable your solution to download them from a public NuGet feed.

Automated Testing and Lab Management Even though TFS Preview doesn't currently support Visual Studio Lab Management, you can still create and run several types of automated tests with Visual Studio. Here are some examples:

- Unit tests
- Coded UI tests
- Web performance tests

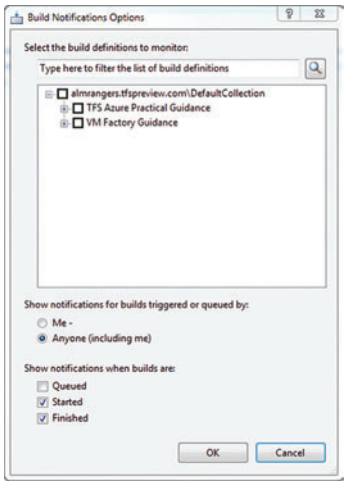


Figure 5 **Monitoring Build Status**

- Load tests
- Generic tests

You can run these tests after a build has completed to evaluate its quality. These tests are often called build verification tests (BVTs) or smoke tests. They typically consist of a broad suite of tests that are used to verify key areas of an application in a particular build. A build is considered a success if all the tests in the BVT pass.

The online MSDN Library article, "How to: Configure and Run Scheduled Tests After Building Your Application," at bit.ly/P2EKB7 describes all the procedures that are required to create and run BVTs.

Doing Coded UI Testing in a Team Build You can use a coded UI test to spice up your BVTs. A coded UI test starts an application as part of the test execution. You must make sure to configure the service account for the build service that's used to start the application. This service account has to be the same as the user account of the active user on this computer. If the service account and the user account aren't the same, the application won't start. You also need to ensure that the Build Service Host property Run the Build Service As is set to Interactive Process in the Team Foundation Server Administration Console/Team Foundation Build Configuration.

Looking Ahead

In future articles, we'll provide guidance on how to set up and configure TFS Preview to enable collaboration for enterprise development shops that want to leverage the services of remote development shops as part of an outsourcing model. For more information on the ALM Rangers, see bit.ly/HrYqAl. ■

Mario Contreras *is a Toronto-based senior consultant with Microsoft Consulting Services, an ALM coach and a frequent speaker at technical conferences. During his tenure at Microsoft Canada, he has led some of the largest Team Foundation Server implementations. Contreras is an active member of the Visual Studio ALM Rangers team.*

Tim Omta *is a senior application development manager in Phoenix. He has worked for Microsoft as both a product developer and a test developer and is currently a field consultant through Microsoft Premier Support for Developers. Omta is an active member of the ALM Rangers.*

Tim Star *is a principal consultant with Intertech Inc., focusing on training, consulting and Visual Studio ALM. He is an MCPD, MCTS, MCT, Visual Studio ALM External Ranger and three-time MVP award recipient. He spends his time writing code, teaching and delivering ALM guidance.*

Thanks to the following technical experts for reviewing this article: *Jim Lamb, Mario Rodriguez, Willy-Peter Schaub and Patricia Wagner*

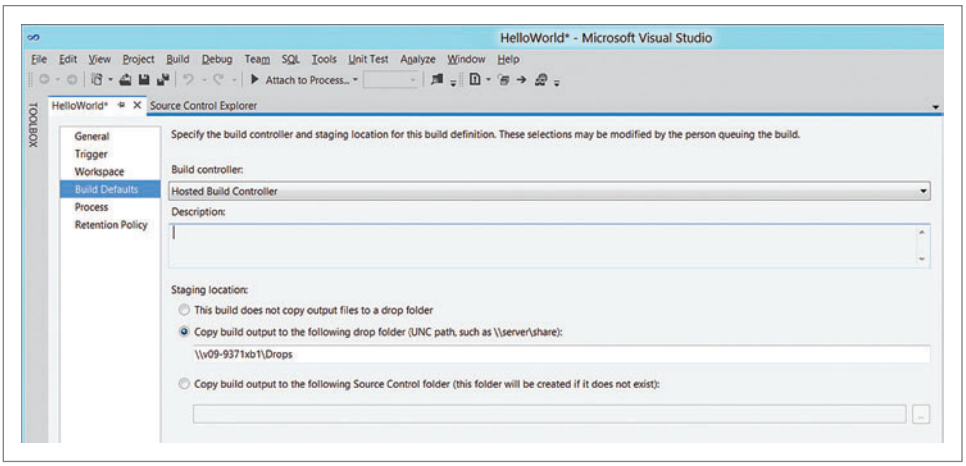


Figure 6 **Selecting a Hosted Build Controller**

TED NEWARD

Cassandra NoSQL Database, Part 3: Clustering

Last time, I examined Apache Cassandra, the "open source, distributed, decentralized, elastically scalable, highly available, fault-tolerant, tuneably consistent, column-oriented database that bases its distribution design on Amazon Dynamo and its data model on Google Bigtable," as described in the book, "Cassandra: The Definitive Guide" (O'Reilly Media, 2010). To be more precise, having installed Cassandra (in the first part in this series), I looked at how to program to it from the Microsoft .NET Framework, doing the basic bits of reading and writing data to it. Nothing spectacular.

The easiest way to set up a Cassandra cluster is to have multiple machines, and obviously one way to do that on a single laptop is to set up multiple virtual machine instances.

In fact, part of Cassandra's "spectacular" is wrapped up in its inherent abilities to cluster well, giving Cassandra easy scale-out. This means it can grow out to "ridiculous" sizes—in most cases with little to no administrative effort—particularly when compared against the work required by most relational databases to store equivalent sizes. As an example, a local tech firm here in Redmond, Wash. (where I live), claimed at a recent startup meetup that it was storing more than 50PB of data in Cassandra.

Even allowing for exaggeration and hyperbole, just one-tenth of that (5PB, or more than 5,000TB) is a pretty hefty chunk of data. To be fair, the Cassandra Web site (cassandra.apache.org) states, "The largest known Cassandra cluster has over 300 terabytes of data in over 400 machines," which is still pretty hard to do with an out-of-the-box relational setup.

But the key to all that storage is in the cluster, and while getting a cluster of that size set up in production is probably beyond the scope of this article, we can at least start to play with it by getting a multi-node cluster running for development work. It requires a few steps, so I'll walk through it one step at a time. (By the way, DataStax has an easy install for Cassandra, but as near as I can tell it lacks the ability to set up a multi-node cluster on one box; that's about its only downside that I can see thus far.)

Install Recap

In the first article in this series (msdn.microsoft.com/magazine/jj553519), I went through the (sometimes agonizing) pain of setting up Cassandra from the .zip file and the command line: Make sure a Java runtime is installed and on the PATH; make sure a JAVA_HOME environment variable is configured; unzip the Cassandra distribution into a directory; and then launch the "cassandra.bat" file from the "bin" directory to get the server up and running.

At the time, it may have seemed really anachronistic to do so, but two positive things come from doing the install that way. First, you get some experience in how to install a server written in Java (and that turns out to be a pretty useful skill to have, given how many of the different NoSQL implementations are written in Java). Second, you'll need to "trick out" that setup at a pretty low level to get Cassandra running multiple times on a single box.

You see, Cassandra's notion of scalability comes from a "ring" of servers: multiple instances of the Cassandra service running on several boxes, each one storing a portion of the total data set. Then, when new data is written to the ring, Cassandra "gossips" (that's the actual technical term for it) between the different nodes in the ring to put the data in the right place within the ring. In a well-administered ring, Cassandra will balance the data between the nodes evenly. Cassandra has a number of different strategies for writing the data out between the nodes, and it's always possible to write a new custom strategy (assuming you're comfortable writing Java), but for now I'm going to stick with the defaults to keep things easier.

One Ring to Rule Them All ...

Normally, the easiest way to set up a Cassandra cluster is to have multiple machines, and obviously one way to do that on a single laptop is to set up multiple virtual machine instances all running simultaneously. But that can get unwieldy and amp up the hardware requirements pretty quickly, particularly if you're one of those developers who does everything off a laptop (like me).

Thus, the second way to get multiple nodes is to have Cassandra run multiple times on the same box, storing data in multiple places and listening on different socket ports. This means diving into

Cassandra's configuration files to set up two (or more) different configuration setups, and launching each.

Assuming a Cassandra 1.1 install (the latest version as of this writing), Cassandra stores all her configuration information in the /conf directory. Within that directory, there are two files in particular I need to edit: log4j-server.properties and cassandra.yaml. I also need to figure out where the nodes' data and logs are going to go, so I'll go ahead and just create two subdirectories under the Cassandra install directory. Assuming you installed Cassandra at C:\Prg\apache-cassandra-1.1.0 (as I did), then you want to create two new directories underneath that, one for each node you're going to create: C:\Prg\apache-cassandra-1.1.0\node1 and \node2.

Within those two directories, copy over the contents of the Cassandra /conf directory, which will bring over those two files you need. You also want to copy over the cassandra.bat file from /bin, because that's where the third and final change will need to happen, in order to tell Cassandra where the configuration files she needs to run will be.

Isn't this Java stuff fun?

The first file, log4j-server.properties, is a configuration file for the log4j diagnostic logging open source project. (Java uses ".properties" files much like Windows used ".ini" files back in the day.) Your main interest here is to make sure that each Cassandra node is writing a diagnostic log file to a different place than the other nodes. Personally, I want all data for each node to be within those \node1 and \node2 directories, so I want to find the line inside log4j-server.properties that reads like this:

```
log4j.appender.R.File=/var/log/cassandra/system.log
```

Then I want to change it to read something more Windows-ish and more \node1-ish, like this:

```
log4j.appender.R.File=C:/Prg/apache-cassandra-1.1.0/node1/log/system.log
```

The log directory doesn't have to exist before Cassandra starts—she'll create it if it isn't there. By the way, make sure the slashes are forward slashes here Just trust me on this one; it'll work. (Java recognizes them whether they're forward or backward slashes, but the properties file syntax uses backward slashes as escape sequence characters, sort of like how they work in C# strings.)

Second, you need to crack open the "cassandra.yaml" file to make the next set of changes. The ".yaml" syntax is "Yet Another Markup Language," and—yes, you guessed it—it's another .ini-style configuration syntax. Java never standardized on this, so it's quite common to see several different configuration styles all conjoined together in a single project (as in Cassandra).

Specifically, you need to change a couple of settings in here; these are scattered throughout the file (which, by the way, is littered with tons of comments, so they're really somewhat self-explanatory if you read through it all):

```
cluster_name: 'Test Cluster'
data_file_directories:
  - /var/lib/cassandra/data
commitlog_directory: /var/lib/cassandra/commitlog
saved_caches_directory: /var/lib/cassandra/saved_caches
listen_address: localhost
rpc_address: localhost
```

The "cluster_name" is optional, but it's not a bad thing to change anyway, maybe to something such as "MyCluster" or "Big Cluster O Fun." The rest of the settings, however, need to be changed. The "directories" entries need to point to the \node1 and \node2 directories, respectively.

One Ring to Find Them All ...

The last two settings need to be changed for different reasons. Cassandra, remember, instinctively wants to run as one service per machine, so she assumes that it's OK to just bind a TCP/IP socket to "localhost." But if you have two or more services running on the same box, that's not going to work. So you need to tell her to bind to addresses that will effectively resolve to the same box, even though they might be different values. Fortunately, you can do this by explicitly putting 127.0.0.1 for node1, 127.0.0.2 for node2 and so forth.

(You might be asking why this works; the answer is beyond the scope of this article, but any good TCP/IP reference should be able to explain it. If you're not convinced, try "ping 127.0.0.1" and "ping 127.0.0.2" on your box. Both should resolve just fine. If you don't like specifying those values, you can always assign them names in your "hosts" file in the C:\Windows\System32\drivers\etc directory.)

> Cassandra needs to be told when she starts up where to find the configuration files.

Part of the reason that Cassandra needs this network configuration worked out is because she's going to "discover" the ring by first connecting to a "seed" node, which will then tell that instance about the other nodes in the ring. This is all part of the gossip protocol that she uses to convey important information around the ring. If we were setting up the ring to run across different machines, Cassandra would need the "seeds" configuration setting to point to a running node, but in this case—because we're all running on the same box—the default 127.0.0.1 works out just fine.

After all the changes, the cassandra.yaml file in \node1 should look like this:

```
cluster_name: 'Test Cluster'
data_file_directories:
  - C:/Prg/apache-cassandra-1.1.0/node1/data
commitlog_directory: C:/Prg/apache-cassandra-1.1.0/node1/commitlog
saved_caches_directory: C:/Prg/apache-cassandra-1.1.0/node1/saved_caches
listen_address: localhost
rpc_address: localhost
```

For \node2, the file should look like this:

```
cluster_name: 'Test Cluster'
data_file_directories:
  - C:/Prg/apache-cassandra-1.1.0/node2/data
commitlog_directory: C:/Prg/apache-cassandra-1.1.0/node2/commitlog
saved_caches_directory: C:/Prg/apache-cassandra-1.1.0/node2/saved_caches
listen_address: 127.0.0.2
rpc_address: 127.0.0.2
```

Finally, Cassandra needs to be told when she starts up where to find the configuration files, and normally she does this by looking along the Java CLASSPATH (which is vaguely similar to the assembly resolution mechanism in the .NET Framework, but about a half-decade more primitive, to be blunt). She also wants to expose some management and monitoring information to JMX (the Java equivalent to PerfMon or Windows Management

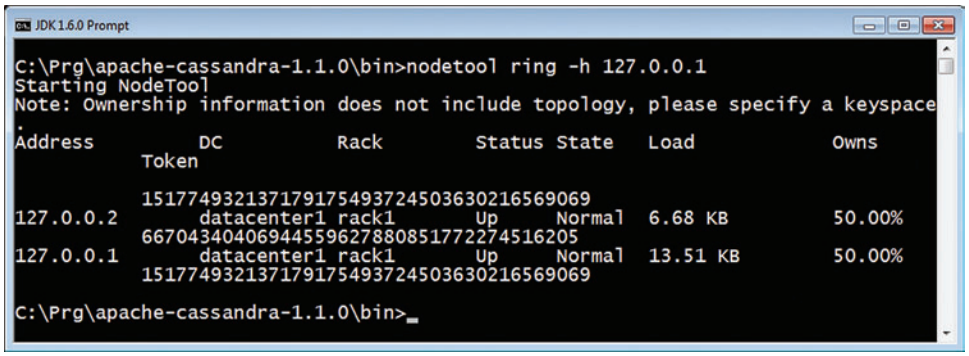


Figure 1 **Two Cassandra Instances, Each Owning 50 Percent of the Data**

Instrumentation) over a TCP/IP port, and both services can't use the same port. Thus, the final changes have to be to cassandra.bat:

```
REM Ensure that any user defined CLASSPATH variables are not used on startup
set CLASSPATH="%CASSANDRA_HOME%\node1"
```

And for the cassandra.bat in \node2:

```
REM Ensure that any user defined CLASSPATH variables are not used on startup
set CLASSPATH="%CASSANDRA_HOME%\node2"
```

As well as the following line in \node2:

```
-Dcom.sun.management.jmxremote.port=7299^
```

In the original, the port will read "7199."

Like I said, isn't this Java stuff fun?

> When used intelligently, it's a powerful new tool in the toolbox, and developers would be foolish to ignore it.

... And in the Darkness Bind Them

But once all the configuration stuff gets out of the way, the fun starts. Fire up a command-prompt window (one with JAVA_HOME and CASSANDRA_HOME environment variables pointing to the root of the JDK and Cassandra install directories, remember), and change directory over to the \node1 directory you've been tricking out. Fire off "cassandra -f" at the prompt, and watch the diagnostic info scroll by. This is the first instance, and assuming all the configuration settings are good (no typos), you should see the text scroll by and end with "Listening for thrift clients ..."

Now, in a second command-prompt window, navigate over to \node2 and do the same thing. This time, as it fires up, you'll also see some activity happen in a few minutes in the \node1 window—what's happening there is that after the \node2 instance gets up and running, it connects to the \node1 instance (the "seed"), and the two essentially configure each other to start working in a ring together. In particular, look for the two lines "JOINING: waiting for ring and schema information" and "Node /127.0.0.1 is now part of the cluster" to appear in the \node2 window, and "Node /127.0.0.2 is now part of the cluster" and "InetAddress /127.0.0.2 is now UP" in the \node1 window.

But, if you missed seeing those messages, Cassandra has one more surprise in store for you. In a third command-prompt window, go to the original Cassandra \bin directory and launch "nodetool ring –h 127.0.0.1," and you should see something like **Figure 1**.

This is really exciting stuff, because as you can see from the Owns column, the two Cassandra instances have already figured out that each one should own 50 percent of the data, without any additional configuration work on your part. Sweet!

The best part is, if you run the code from the previous article, the data will be spread across the cluster without any additional changes.

It's a Complement, Not a Replacement

Like some of the other database tools this column has explored (MongoDB and SQLite), Cassandra shouldn't be considered as a wholesale replacement for a relational database, but as a complementary technology that can be used either for areas where the feature set of a relational database just doesn't fit well (caching or storing highly unstructured data sets come to mind, for example), or as a hybrid system, in conjunction with a relational database. For example, a company might store a "fixed" set of data elements in a relational database and include as one of the relational columns a Cassandra key, in order to retrieve the remaining, unstructured data. The relational database can then remain structured and relational (obeying most or all of the normal-form rules), but the system overall will still have the flexibility to store additional unanticipated data elements that users always seem to want to add to the system as it ages.

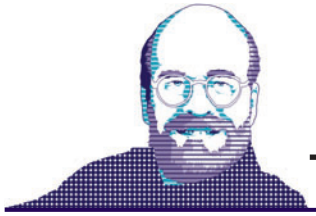
For another example, consider Web page hit data, which would always be keyed off the page itself, yet would easily track into the millions or billions of elements of data. A URL-shortening service (such as bit.ly) would be trivial to do here, because the minimized URL path (the "foobar" part in http://bit.ly/foobar) would be the key, and hit data stats—as well as an optional description and even perhaps a periodic snapshot of the redirected URL—would be made for Cassandra. And so on.

Cassandra isn't going to take over the datacenter anytime soon, nor should it. But when used intelligently, it's a powerful new tool in the toolbox, and developers would be foolish to ignore it. There's a lot more to explore about Cassandra, but it's time to let the Trojan prophetess go and move on to other things.

Happy coding! ■

Ted Neward *is an architectural consultant with Neudesic LLC. He has written more than 100 articles and authored or coauthored a dozen books, including "Professional F# 2.0" (Wrox, 2010). He's an F# MVP and noted Java expert, and speaks at both Java and .NET conferences around the world. He consults and mentors regularly—reach him at ted@tedneward.com or Ted.Neward@neudesic.com if you'd like him to come work with your team. He blogs at blogs.tedneward.com and can be followed on Twitter at twitter.com/tedneward.*

Thanks to the following technical expert for reviewing this article: *Kelly Sommers*



CHARLES PETZOLD

Assembling Bing Map Tiles on Windows Phone

The Motion sensor in Windows Phone consolidates information from the phone's compass and accelerometer to create a rotation matrix that describes the orientation of the phone in 3D space. Recently I began pondering how the phone's orientation could be used in combination with Bing Maps. I anticipated a quickie mashup, but the job turned out to be rather more complex.

If you've experimented with the standard Maps program on your Windows Phone, you know that the display is usually aligned so that north is toward the top of the phone. (The only exception is when you're using the map for directions to a location, in which case the map is oriented to indicate the direction you're travelling.) Up for north is the convention for maps, of course, but in some cases you might want the phone's map to rotate relative to the phone so that north on the map is actually pointing north.

It seems so simple, right?

Map Control Limitations

In my quest to implement a rotating map on the phone, I started with the Bing Maps Silverlight Control for Windows Phone, the center of which is a control named simply Map in the Microsoft.Phone.Controls.Maps namespace.

> In some cases you might want the phone's map to rotate relative to the phone so that north on the map is actually pointing north.

To get started with the Map control (or anything involving accessing Bing Maps programmatically) you need to register at the Bing Maps Account Center at bingmapsportal.com. It's a straightforward process to obtain a credentials key that gives your program access to Bing Maps.

Code download available at archive.msdn.microsoft.com/mag201211TouchAndGo.

In the Windows Phone project that uses the Map control, you'll need a reference to the Microsoft.Phone.Controls.Maps assembly, and you'll probably want an XML namespace declaration in the XAML file (in one line):

```
xmlns:maps="clr-namespace:Microsoft.Phone.Controls.Maps;
  assembly=Microsoft.Phone.Controls.Maps"
```

Instantiating a basic map is then trivial:

```
<maps:Map CredentialsProvider="credentials-key" />
```

Insert the actual credentials key you've obtained from the Bing Maps Account Center.

Yes, getting a map on the screen is easy, but when I searched out ways to rotate it, I came up dry. For sure, the Map class inherits a Heading property from the MapBase class, but apparently this property is relevant only for "bird's eye" maps, and the Map control supports only the standard road and aerial views.

Of course, it's fairly trivial to rotate the Map control by setting a RotateTransform object to its RenderTransform property, but I wasn't happy at all with that solution. For one thing, it tended to obscure the copyright notice at the bottom of the map, and doing that seemed to be in violation of the conditions I agreed to when obtaining a credentials key.

I decided to abandon the Maps control and instead try my luck on a much lower level by accessing the Bing Maps SOAP Services. This set of Web services allows a program to obtain the actual bitmap tiles from which larger maps are constructed.

Figure 1 **Code to Access the Bing Maps Imagery Web Service**

```
void OnMainPageLoaded(object sender, RoutedEventArgs e)
{
  // Initialize the Bing Maps imagery service
  ImageryServiceClient imageryServiceClient =
    new ImageryServiceClient("BasicHttpBinding_IImageryService");
    imageryServiceClient.GetImageryMetadataCompleted +=
      GetImageryMetadataCompleted;

  // Make a request for the road metadata
  ImageryMetadataRequest request = new ImageryMetadataRequest
  {
    Credentials = new Credentials
    {
      ApplicationId = "credentials-key"
    },
    Style = MapStyle.Road
  };
  imageryServiceClient.GetImageryMetadataAsync(request, "road");

  // Make a request for the aerial metadata
  request.Style = MapStyle.Aerial;
  imageryServiceClient.GetImageryMetadataAsync(request, "aerial");
}
```

Figure 2 **The Completed Handler for the Bing Maps Web Service**

```
void GetImageryMetadataCompleted(object sender,
                                 GetImageryMetadataCompletedEventArgs args)
{
  if (!args.Cancelled && args.Error == null)
  {
    // Get the "powered by" bitmap
    poweredByBitmap.UriSource = args.Result.BrandLogoUri;
    poweredByDisplay.Visibility = Visibility.Visible;

    // Get the range of map levels available
    ImageryMetadataResult result = args.Result.Results[0];
    minimumLevel = result.ZoomRange.From;
    maximumLevel = result.ZoomRange.To;

    // Get the URI and make some substitutions
    string uri = result.ImageUri;
    uri = uri.Replace("{subdomain}", result.ImageUriSubdomains[0]);
    uri = uri.Replace("&token={token}", "");
    uri = uri.Replace("{culture}", "en-US");

    if (args.UserState as string == "road")
      roadUriTemplate = uri;
    else
      aerialUriTemplate = uri;

    if (roadUriTemplate != null && aerialUriTemplate != null)
      RefreshDisplay();
  }
  else
  {
    errorTextBlock.Text =
      "Cannot access Bing Maps: " + args.Error.Message;
  }
}
```

Accessing the SOAP Service

In a Web service built around Simple Object Access Protocol (SOAP), information is transferred between your program and the server using XML documents. Very often these documents involve rather complex data structures, so instead of handling the XML directly, a much easier approach is to have Visual Studio create a proxy class for you. This allows your program to access the Web service with normal (albeit asynchronous) C# classes and method calls.

The Bing Maps SOAP Services interface is documented at bit.ly/S3R4IG, and consists of four separate services:

- **Geocode Service:** Match addresses with longitude and latitude.
- **Imagery Service:** Obtain maps and tiles.
- **Route Service:** Get directions.
- **Search Service:** Locate people and businesses.

I was only interested in the Imagery Service.

The downloadable code for this article is a single project named RotatingMapTiles. I added a proxy for the Imagery Service to this project by choosing Add Service Reference from the Project menu. In the Address field of the Add Service Reference dialog, I copied the URL listed in the Bing Maps SOAP Services Addresses section of the documentation and pressed Go. When the service was located, I gave it a name of ImageryService in the Namespace field.

The C# code generated for this service has a namespace that's the normal project namespace, followed by the namespace you specify when creating the service reference, so the MainPage.xaml.cs file in the RotatingMapTile program contains the following using directive:

```
using RotatingMapTiles.ImageryService;
```

Figure 3 **The Four Level 1 Tiles**

Figure 4 **The 16 Level 2 Tiles**

The tiles that form the basis of Bing Maps are bitmaps that are always 256 pixels square.

The Imagery Service supports two types of requests involving the function calls GetMapUriAsync and GetImageryMetadataAsync. The first call allows you to obtain static maps of a particular location, size and zoom level, while the second works at a lower level. Among the metadata this second call returns is a URI template that lets you access the actual bitmap tiles that are used to assemble all the Bing maps.

Figure 1 shows the Loaded handler for the MainPage class in the RotatingMapTiles project making two calls to the Imagery Web service to obtain metadata for the MapStyle.Road and MapStyle.Aerial styles. (MapStyle.Birdseye is also available but it's rather more complex to use.)

You'll need your own Bing Maps credential key to substitute for the placeholder.

Figure 2 shows the handler for the Completed event of the asynchronous call. The information includes a URI for a bitmap with the Bing logo, so it's easy to credit Bing Maps on the program's screen.

The other URI is the one you'll use to access the map tiles. There are separate URIs for the road and aerial views, and the URI contains a placeholder for a number that identifies precisely the tile you want.

Maps and Tiles

The tiles that form the basis of Bing Maps are bitmaps that are always 256 pixels square. Each tile is associated with a particular longitude, latitude and zoom level, and contains an image of a square area on the surface of the Earth flattened using the common Mercator projection.

The most extreme zoomed-out view is known as Level 1, and only four tiles are required to cover the entire world—or at least the part of the world with latitudes between positive and negative 85.05°—as shown in **Figure 3**.

I'll explain the numbers on the tiles in a moment. Because the tiles are 256 pixels square, at the equator each pixel is equivalent to about 49 miles.

Level 2 is more granular, and now 16 tiles cover the Earth, as shown in **Figure 4**.

The tiles in **Figure 4** are also 256 pixels square, so at the equator each pixel is about 24 miles. Notice that each tile in Level 1 covers the same area as 4 tiles in Level 2.

Figure 5 **A Routine to Calculate a Quadkey**

```
string ToQuadKey(int longitude, int latitude, int level)
{
  long quadkey = 0;
  int mask = 1 << (level - 1);

  for (int i = 0; i < level; i++)
  {
    quadkey <<= 2;

    if ((longitude & mask) != 0)
      quadkey |= 1;

    if ((latitude & mask) != 0)
      quadkey |= 2;

    mask >>= 1;
  }

  strBuilder.Clear();

  for (int i = 0; i < level; i++)
  {
    strBuilder.Insert(0, (quadkey & 3).ToString());
    quadkey >>= 2;
  }

  return strBuilder.ToString();
}
```

This scheme continues: Level 3 has 64 tiles, Level 4 has 256 tiles, and up and up and up to Level 21, which covers the Earth with a total of more than 4 trillion tiles—2 million horizontally and 2 million vertically for a resolution (at the equator) of 3 inches per pixel.

Numbering the Tiles

Because at least a few of these trillions of tiles must be individually referenced by a program that wishes to use them, they must be identified in a clear and consistent manner. There are three dimensions involved—longitude, latitude and zoom level—and a practical consideration as well: To minimize disk accesses on the server, tiles associated with the same area should be stored near each other, which implies a single numbering system that encompasses all three dimensions in some very clever manner.

The clever numbering system used for these map tiles is called a "quadkey." The MSDN Library article, "Bing Maps Tile System," by Joe Schwartz (bit.ly/SxVojl) is a really good explanation of the system (including helpful code) but I'll take a somewhat different approach here.

Each tile has a unique quadkey. The tile URIs obtained from the Web service contain a placeholder string "{quadkey}". Before you use one of the URIs to access a tile, you must replace this placeholder with an actual quadkey.

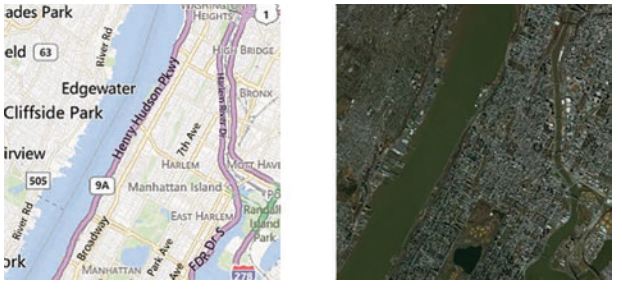


Figure 6 **The Tiles for Quadkey "032010110130"**

Figure 3 and **Figure 4** show the quadkeys for zoom Levels 1 and 2. Leading zeros are important in quadkeys. (Indeed, you might want to think of the quadkey as a string rather than a number.) The number of digits in a quadkey is always equal to the zoom level of the tile. The tiles in Level 21 are identified with 21-digit quadkeys.

The individual digits of a quadkey are always 0, 1, 2 or 3. Thus, the quadkey is really a base-4 number. Look at these four digits in binary (00, 01, 10, 11) and how they appear in a group of four tiles. In each base-4 digit, the second bit is really a horizontal coordinate and the first bit is a vertical coordinate. The bits correspond to a longitude and latitude, which are effectively interleaved in the quadkey.

Each tile in Level 1 covers the same area as a group of four tiles in Level 2. You can think of a tile in Level 1 as a "parent" to four "children" in Level 2. The quadkey of a child tile always begins with the same digits as its parent, and then adds another digit (0, 1, 2 or 3) depending on its location within the area of its parent. Going from parent to child is a zoom up. Zooming down is similar: For

Figure 7 **The RefreshDisplay Method in RotatingMapTiles**

```
void RefreshDisplay()
{
  if (roadUriTemplate == null || aerialUriTemplate == null)
    return;

  if (integerLongitude == -1 || integerLatitude == -1)
    return;

  // Get coordinates and pixel offsets based on current zoom level
  int croppedLongitude = integerLongitude >> BITRES - zoomLevel;
  int croppedLatitude = integerLatitude >> BITRES - zoomLevel;
  int xPixelOffset = (integerLongitude >> BITRES - zoomLevel - 8) % 256;
  int yPixelOffset = (integerLatitude >> BITRES - zoomLevel - 8) % 256;

  // Prepare for the loop
  string uriTemplate = mapStyle ==
    MapStyle.Road ? roadUriTemplate : aerialUriTemplate;
  int index = 0;
  int maxValue = (1 << zoomLevel) - 1;

  // Loop through the 5x5 array of Image elements
  for (int row = -2; row <= 2; row++)
    for (int col = -2; col <= 2; col++)
    {
      // Get the Image and BitmapImage
      Image image = imageCanvas.Children[index] as Image;
      BitmapImage bitmap = image.Source as BitmapImage;
      index++;

      // Check if you've gone beyond the bounds
      if (croppedLongitude + col < 0 ||
        croppedLongitude + col > maxValue ||
        croppedLatitude + row < 0 ||
        croppedLatitude + row > maxValue)
      {
        bitmap.UriSource = null;
      }
      else
      {
        // Calculate a quadkey and set URI to bitmap
        int longitude = croppedLongitude + col;
        int latitude = croppedLatitude + row;
        string strQuadkey =
          ToQuadKey(longitude, latitude, zoomLevel);
        string uri = uriTemplate.Replace("{quadkey}", strQuadkey);
        bitmap.UriSource = new Uri(uri);
      }

      // Position the Image element
      Canvas.SetLeft(image, col * 256 - xPixelOffset);
      Canvas.SetTop(image, row * 256 - yPixelOffset);
    }
}
```

any child quadkey, you obtain the parent quadkey simply by lopping off the last digit.

Here's how to derive a quadkey from an actual geographic longitude and latitude.

Longitude ranges from -180° at the International Date Line, and then increases going east to 180° at the International Data Line again. For any longitude, first calculate a relative longitude that ranges from 0 to 1 with 0.5 representing the prime meridian:

```
double relativeLongitude = (180 + longitude) / 360;
```

Now convert that to an integer of a fixed number of bits:

```
int integerLongitude =
  (int)(relativeLongitude * (1 << BITRES));
```

In my program I've set BITRES to 29 for the 21 zoom levels plus 8 bits for the pixel size of the tile. Thus, this integer identifies a longitude precise to the nearest pixel of a tile at the highest zoom level.

The calculation of integerLatitude is a little more complex because the Mercator map projection compresses latitudes as you get further from the equator:

```
double sinTerm = Math.Sin(Math.PI * latitude / 180);
double relativeLatitude =
  0.5 - Math.Log((1 + sinTerm) / (1 - sinTerm))
    / (4 * Math.PI);
int integerLatitude = (int)(relativeLatitude * (1 << BITRES));
```

The integerLatitude ranges from 0 at 85.05° north of the equator to the maximum value at 85.05° south of the equator.

The center of Central Park in New York City has a longitude of -73.965368° and a latitude of 40.783271°. The relative values are (to just a few decimal places) 0.29454 and 0.37572. The 29-bit integer longitude and latitude values (shown in binary and grouped for easier readability) are:

```
0 1001 0110 1100 1110 0000 1000 0000
0 1100 0000 0101 1110 1011 0000 0000
```

Suppose you want a tile that shows the center of Central Park in a Level 12 zoom. Take the top 12 bits of the integer longitudes and latitudes (watch out—the following digits are grouped a little differently than the 29-bit versions):

```
0100 1011 0110
0110 0000 0010
```

These are two binary numbers but we need to combine them to form a base-4 number. There's no way to do this in code using simple arithmetical operators. You need a little routine that actually goes through the individual bits and constructs a longer integer or a string. For illustrative purposes, you can simply double all the bits in the latitude and add the two values as if they were base-4 values:

```
0100 1011 0110
0220 0000 0020
0320 1011 0130
```

The result is the 12-digit quadkey you'll need to substitute for the "{quadkey}" placeholder in the URI you get from the Web service to access the map tile.

Figure 5 shows a routine to construct a quadkey from the truncated integer longitudes and latitudes. For clarity, I've separated the logic into sections that generate a quadkey long integer and a quadkey string.

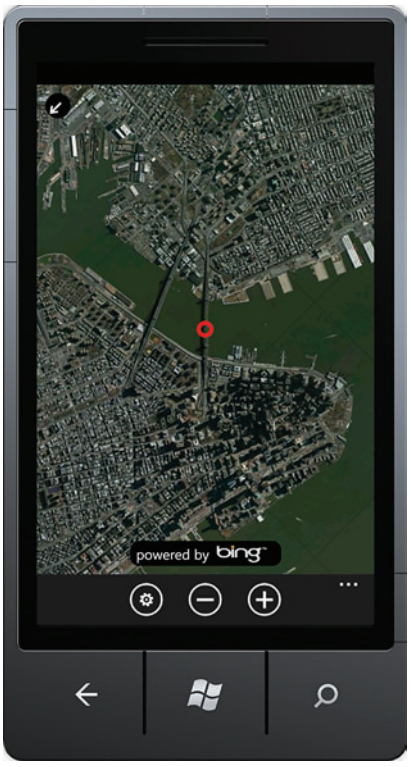


Figure 8 **The RotatingMapTiles Display**

Figure 6 shows both the road and aerial tiles for this quadkey. The center of Central Park is actually way down at the bottom of these images, a bit to the left of center. This is predictable from the integer longitudes and latitudes. Look at the next 8 bits of the integer longitude after the first 12 bits: The bits are 0111 0000 or 112. The next 8 bits of the latitude are 1111 0101 or 245. This means the center of Central Park is the 112th pixel from the left and the 245th pixel down in those tiles.

Tiling the Tiles

Once you've truncated an integer longitude and latitude to a certain number of bits corresponding to a particular zoom level, obtaining adjacent tiles is a snap: Simply increment and decrement the longitude and latitude integers and form new quadkeys.

You've already seen three methods from the MainPage class of the RotatingMapTiles project. The program uses a GeoCoordinateWatcher to obtain the longitude and latitude of the phone, and converts the coordinates to integer values as shown earlier. The application bar has three buttons: to toggle between road and aerial views and to increase and decrease the zoom level.

The program has no other touch interface besides the buttons. It always displays the location obtained from the GeoCoordinateWatcher in the center of the screen and constructs the total map with 25 Image elements in a 5x5 array, a configuration that always fills the 480x800 pixel screen, even with rotation. The MainPage class creates these 25 Image elements and BitmapImage objects in its constructor.

Whenever the GeoCoordinateWatcher comes up with a new location, or the zoom level or map style changes, the RefreshDisplay method in **Figure 7** is called. This method shows how the new URIs are obtained and simply set to the existing BitmapImage objects.

To keep this program reasonably simple, it doesn't attempt to smooth over the transitions between views and zoom levels. Often the whole screen goes blank as new tiles are being loaded.

But the program does rotate the map. The rotation logic is based on the Motion sensor and a RotateTransform and is pretty much independent of the rest of the program. **Figure 8** shows me taking my Windows Phone (or perhaps the Windows Phone emulator) for a walk across the Brooklyn Bridge. The top of the phone is pointed in the direction I'm walking, and the little arrow in the upper-left corner indicates north. ■

Charles Petzold *is a longtime contributor to* MSDN Magazine, *and the author of "Programming Windows, 6th edition" (O'Reilly Media, 2012), a book about writing applications for Windows 8. His Web site is charlespetzold.com.*

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

DAVID S. PLATT

Here We Go Again

Brace yourselves. Microsoft is launching a new graphical environment with Windows 8. The new UI offers powerful capabilities such as a native touch interface and interactive Live Tiles. But it also means that we're about to be deluged with awful UXs built by alleged professionals whose job it is to know better and do better.

Every time a new graphical environment arrives, users pay the price while designers relearn the basic principles that should be branded on their hearts: Users don't care about your app for itself—they only care about their own problems and their own pleasures.

Designers swoon over the flashy new tools like a 6-year-old who got an Erector Set for Christmas. They throw these new graphical elements into applications almost at random, saying, "Look Ma, see what I can do! Isn't it cool?"

> The magic widget lives in the same place it always has: between your ears.

No. It isn't. Parents finishing their tasks more quickly so they can play with their kids is cool. Grandparents talking to grandchildren on a video-conference app that neither *notices at all* is cool. Your flashing video buttons screaming, "BOOYAH! LOOK AT ME!" are not cool.

I've seen this happen more times than I care to remember. Windows inspired a parade of nonsensical menu item names and inconsistent menu options. HTML brought convoluted navigation structures and a carnival of scrolling marquees and spinning logos that delighted the *marketingbozos* at wobbly Internet startups, but nauseated their users. Windows Presentation Foundation (WPF) and Silverlight unleashed color gradients, of which even simple ones, misapplied, can inflict genuine physical pain in less than 30 seconds. See my white paper, "Using WPF for Good and Not Evil" (bit.ly/bkzw5f), for a demonstration.

Now comes the new category of Windows Store apps built on the Windows 8 UI. You'll pardon me if I cringe at the learning curve that I foresee here. Because it *doesn't have to happen* that way.

Microsoft has for years taken a hands-off approach to UX design. The company always provides tons of how-to information for its products, yet it has historically provided little when-to and when-not-to guidance. That's not OK. Having developed a chain saw, Microsoft incurs a duty to teach its customers which end of it to hold.

To its credit, Microsoft has improved in this area. The Windows Dev Center contains a section on "Designing UX for Apps" (bit.ly/yHiblE), for example. It's a decent start, and better than what we had before (which was nothing at all). But so much more needs doing. If Microsoft forcefully leads the way, we'll see faster uptake and fewer cringe-worthy apps. But far too many developers and designers keep saying, "Wow, Microsoft is giving us this magic widget. I'll throw it at my users and they'll love it."

Balderdash. The magic widget that will delight your users is not in your Visual Studio toolbox. Not with Windows, not with HTML, not with WPF or Silverlight, and now not with Windows 8, either. The magic widget lives in the same place it always has: between your ears.

The magic comes from knowing who your users are: what they want, what they need, what they only think they want, and what they haven't yet realized they need. You need to know which problems they're trying to solve and what they would consider the characteristics of a good solution. Only then can you say, "I know who they are and what they need—how close can I get to that with the toolkit I have?"

To take whatever whiz-bangery Microsoft releases and just throw it at a user hoping it will somehow make him happy (because it makes you happy, you hopeless geek) is bad programming. It's lazy programming. To use the most toxic word in the geek vocabulary, it's stupid programming.

It's time for this industry, *our* industry, our *profession*, to grow up. This time, let's start from the user's perspective, not from the toolkit. Then maybe our programs will suck less sooner.

I'll say it again: The magic gizmo isn't on your computer. It's between your ears. Start using it. You're smart enough. Aren't you?

End Note: I feel so strongly about UX that I've affiliated with the company IDesign to better bring this message to the world. It will be marketing my courses on the topic, starting in London on Oct. 8 (bit.ly/Tt2vrp). Be there! Aloha! ■

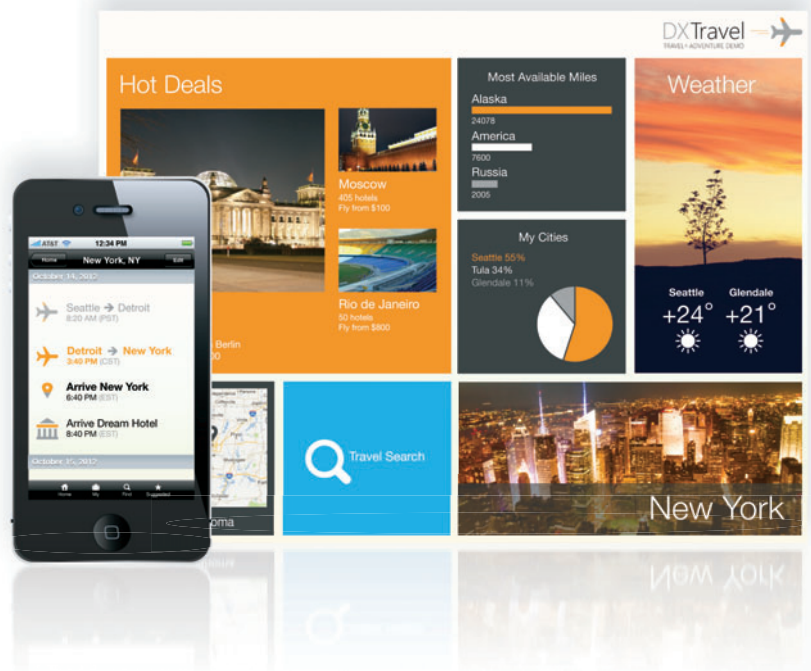
DAVID S. PLATT *teaches programming .NET at Harvard University Extension School and at companies all over the world. He's the author of 11 programming books, including "Why Software Sucks" (Addison-Wesley Professional, 2006) and "Introducing Microsoft .NET" (Microsoft Press, 2002). Microsoft named him 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.*

DevExpress

Let's see what develops.

discover DXTREME.

Delight your users by creating apps that feel as though they were designed expressly for the device. With **DXTREME**, multi-channel means building applications that span devices and optimize the best parts of each platform. And with HTML5/JS visualization built in your dynamic charts and graphs will be both powerful and beautiful.



Easy does it.

Are you ready to go cross-platform?

Download the **DXTREME** Preview to experience the future.

www.DevExpress.com

DXv2

The next generation of inspiring tools. **Today.**

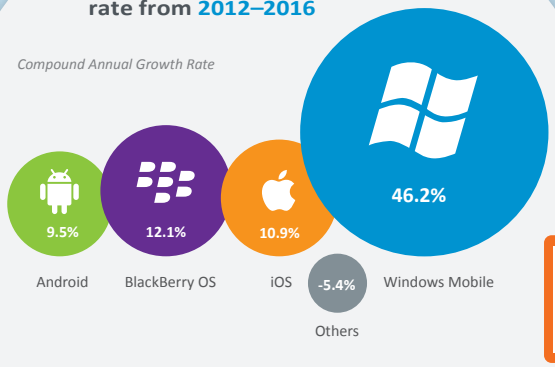
u dxt dxp asp win wpf sl 8

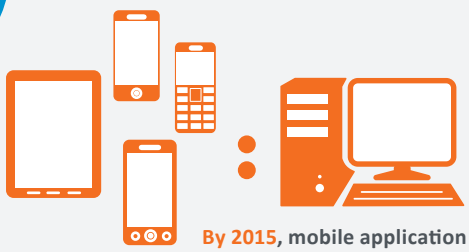
THE FUTURE OF MOBILE DEVELOPMENT

By 2015, there will be **1** mobile phone for every person on Earth.

Worldwide smartphone operating system growth rate from 2012–2016

Compound Annual Growth Rate

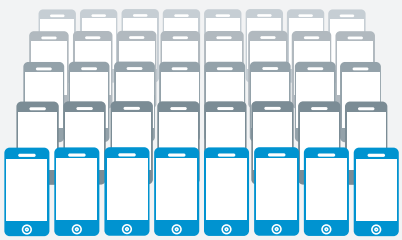




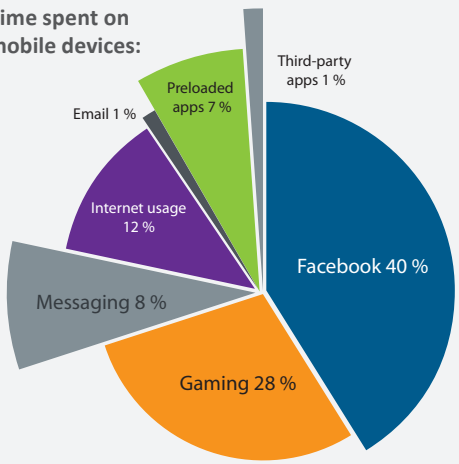
By 2015, mobile application development projects targeting smartphones and tablets will outnumber native PC projects by a **ratio of 4 to 1**.

$17 billion

Analyst firm Forrester Research predicts corporations will spend up to **$17 billion** creating mobile applications for their products and working with third-party services and companies that manage these applications.



Time spent on mobile devices:



Mobile App Downloads

Syncfusion has the answer to your mobile development …

ORUBASE!

Syncfusion Orubase is a complete framework that enables you to develop hybrid mobile applications easier. It provides an extensive set of mobile web controls that can be directly used in your ASP.NET MVC web applications to produce stunning platform-specific user interfaces. Orubase also provides native wrapper frameworks on the iOS, Android, and Windows Phone platforms without requiring third-party wrappers.

Download a free 30-day trial today! www.orubase.com/MSDN

Syncfusion®