*BIZ & IT —*

# OneCore to rule them all: How Windows Everywhere finally happened

Microsoft promised developers that Windows would run anywhere. This summer, it finally will.

ARS STAFF - 5/20/2016, 4:00 AM

## As one fades away, another enters the scene

This brush-clearing made things easier for third-party developers, but it didn't do much to help Microsoft. Internally, the trifurcation soon happened once again thanks to a thing called the Xbox. The original Xbox operating system was a fork of Windows 2000. When Microsoft made the fork, the approach was what Microsoft Distinguished Engineer Don Box called "fork and run." Rather than keeping the Xbox operating system synchronized with the mainstream development of Windows 2000 (and its successors), the Xbox operating system was left free to diverge.

Again, this was a decision that made a lot of sense in the moment. The Xbox operating system is designed to be more predictable than desktop Windows, so for example it lacks most of Windows' demand-based paged virtual memory. In Windows on a PC, the operating system can move the memory used by one application from RAM to the pagefile on disk, thereby freeing up the RAM so that another application can use it. This action comes at the expense of making the first application slow down, as using that data means waiting for it to be read from the disk. This is useful on a PC since it allows many applications to run concurrently even if they use more memory than the system has. But such a setup is bad on a games console; that slow wait for the disk means performance drops, causing frame rates to stutter. So the Xbox operating system drops the capability entirely.

Similarly, Xbox doesn't have any great need to multitask. Turn on an original Xbox and it boots into the Xbox operating system, but as soon as you start a game, the operating system mostly unloads to free up memory for the game itself. Quit the game and the operating system loads back up again. This style of operation (which is oddly similar to the way DOS used to work) made sense in a single-purpose console, but it made no sense in a more general-purpose operating system.

The Xbox 360 operating system didn't do much to change this, either. Though released in 2005, its operating system was not synchronized with Windows XP or Windows Server 2003 (the latest iterations of the Windows NT kernel). Instead, it was a continuation of the Windows 2000-derived Xbox operating system.

FURTHER READING
Xbox 360: the Ars Technica Review

When the Windows Everywhere vision was first articulated, a games console was not on the cards. For much of the Xbox platform's life, it was a special thing all on its own. It happened to run a Windows-derived operating system, but it was never an application platform in the way that Windows 95, Windows NT, and Windows CE were. It wasn't a Win32 platform.

Over on the phone, the wheels were starting to come off Windows CE. After being very *early* to the idea of "a phone that can run applications" with the Windows CE-based Windows Mobile, Microsoft was rather late to "a phone that can run applications *running an operating system that people like to use*." Belatedly realizing that Windows Mobile, with its scaled-down Windows 95 lookalike interface and dependence on itty bitty styluses, was never going to win hearts and minds, Microsoft came up with Windows Phone 7.

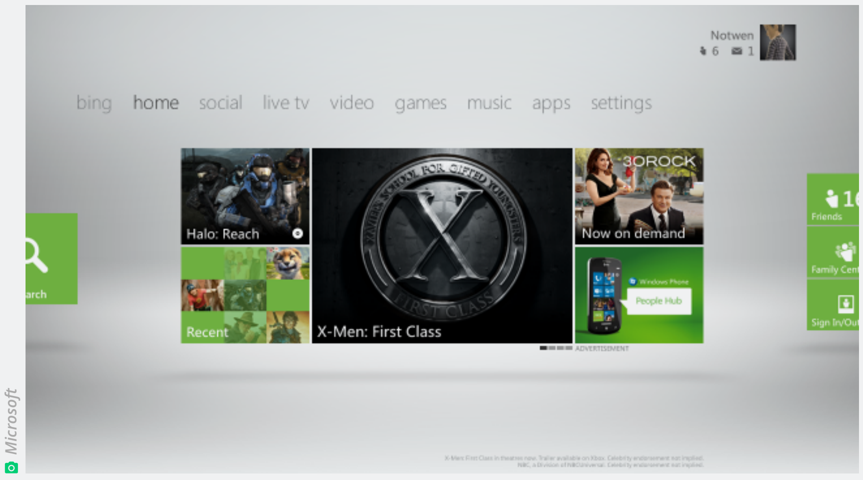
Out of necessity, this was built on top of Windows CE: Windows CE was where Microsoft had a telephony stack, Windows CE ran on ARM processors, and Windows CE worked on small systems with limited RAM and storage. Microsoft needed to get something to market as fast as it could, and it was much easier to use an operating system that already had all these fundamental capabilities rather than starting from scratch or beating Windows NT into shape. As such, Windows Phone 7 hit the market in 2010.

FURTHER READING
Windows Phone 7: The Ars Review

Cognizant of the fact that its latecomer platform had to be developer-friendly, Windows Phone 7 *wasn't* a Win32 platform (at least as far as third-party developers were concerned). Microsoft instead went where most of its third party developers actually were: .NET. The Windows Phone 7 developer model was based, more or less, on Silverlight, the cut-down version of .NET that was once promoted as a competitor to the Flash browser plugin.

As a time-to-market compromise, this was another decision that made sense. But it also proved to be deeply limiting. It wasn't long before dual (or more) core smartphone processors started showing up, which Windows CE didn't support. While Windows CE had improved over the years, moving from the 32MB memory slice model to something more like Windows 95, it was nonetheless limiting. The lack of strong security and multicore support in particular were untenable moving forward. The not-quite-Win32 API meant that it couldn't run the same full Silverlight as found on the desktop, or the full Internet Explorer rendering engine. Windows Phone had rapidly outgrown Windows CE, giving Microsoft a new dilemma. Should it build Windows CE up to make it better placed to handle the new demands placed on it, or should it cut Windows NT down to make it fit onto a phone?

## Apps for Xbox



Enlarge / The Xbox 360's 2011 dashboard update.

Meanwhile, the Xbox team wanted to make the Xbox 360 into something more of an application platform. The Xbox 360 received a major update to its operating system and user interface in late 2011, picking up new streaming media capabilities such as a YouTube app. A second update in 2012 added a Web browser.

In both cases, Microsoft reused its own technology, using Silverlight for apps and Internet Explorer for the browser. The Xbox's forked operating system made this much harder to do, however. Internet Explorer and Silverlight were built to run on real desktop Windows, a Windows that had been under continuous development ever since the Xbox team forked Windows 2000. To get these things onto the Xbox 360, the Xbox team ended up having to fork them and rework them to fit the constraints and capabilities of the games console.

Windows uses. The same was true, perhaps less visibly, of the Silverlight implementation.

Maintaining a forked version of a piece of software takes work: changes coming from the upstream origin have to be ported and, from time to time, fixed to accommodate the differences that the fork imposes. Microsoft hadn't been really paying this price with the core Xbox operating system. Instead, it allowed the two operating systems to become increasingly different. This resulted in a certain amount of parallel development. Windows 2000, for example, didn't natively support Wi-Fi connectivity. That was added in Windows XP. Accordingly, when Wi-Fi was added to the Xbox, the Xbox team couldn't just take the Windows XP Wi-Fi support, because that depended on many of the divergent changes made during Windows XP's development. Instead, they had to develop their own Wi-Fi support just for Xbox OS.

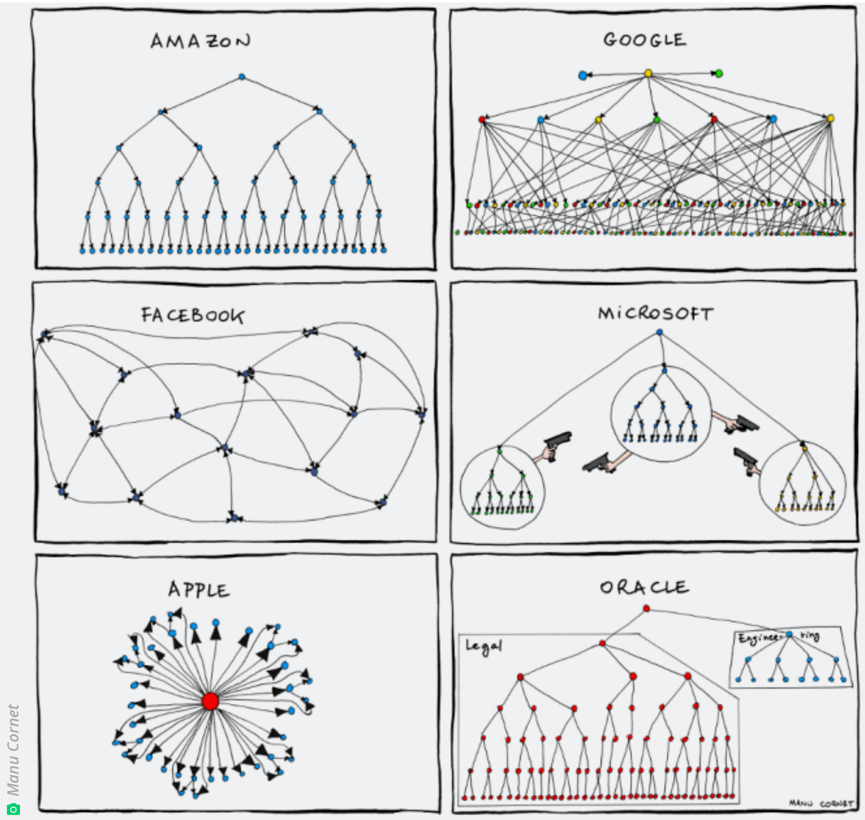
The same parallel development afflicted Windows CE, too. As a separate operating system, it also needed its own Wi-Fi stack. Windows Phone 7 included its own fork of Internet Explorer, and so on and so forth.

This was tolerable for a long time. But by late 2010—after Windows Phone 7 shipped and as the Xbox 360 update was being developed—it became increasingly clear that the cost and complexity was becoming prohibitive: three operating system kernels, three sets of new hardware support, three forked browsers, three versions of Win32, three graphical stacks and versions of DirectX. The overheads add up.

It wasn't much good for third parties, either. Windows Phone app development was quite different from Win32 app development. For the Xbox One, Microsoft had the ambition of opening up its console as an application platform, but the fork and porting experience of bringing Silverlight and Internet Explorer to the Xbox 360 made it clear that the Xbox could no longer stand alone as its own thing. It had to be compatible with other Microsoft platforms, both to offer a consistent developer experience for third parties and to be a tractable maintenance experience for Redmond itself.

As the Phone team was developing Windows Phone 8 in 2011 and 2012, the Xbox team worked on the Xbox One in 2012 and 2013. Soon it became clear to both groups that the triple operating system strategy was not working out. The Phone team needed a modern, fully featured operating system to handle their hardware and application support needs. Windows CE could have probably been stretched to last another generation or two (and in fact, it did gain limited support for multiple cores and lose the 32MB memory slice design), but Windows NT was obviously the more capable, more robust platform to build on. The Xbox team needed to be running mainline Windows so that they could support a rich application ecosystem. The costs and limitations of being independent were becoming prohibitive.

## Meanwhile, in the Windows division....



Enlarge / **The org chart comic isn't exactly right, but it wasn't exactly wrong, either.**

While the Phone and Xbox teams were doing their things, the Windows team was doing its thing. The widely propagated joke of the Microsoft org chart with a bunch of divisions pointing guns at each other encapsulated a certain truth about the company; Phone, Xbox, and Desktop were separate groups, with separate developers, separate responsibilities, and separate goals despite everyone working together for Microsoft. The Windows team wasn't writing an operating system for phones or for games consoles; it was writing one for desktop and server computers.

Fortunately, the Windows team had embarked on a project that would become instrumental in creating a common platform for all these various demands. That project is already widely known—it was called MinWin—if not necessarily widely understood.

Remember that Windows in the server room was once seen as a bold new market for Microsoft. But by the time of Windows Server 2003's release, this focus was well established. With that, however, came an increasing desire to make Windows more Linux-like in many ways. Linux server systems can be pared down to remove unnecessary packages and hence reduce the security surface area; they can also be managed fully from the command line. There were increasingly calls to enable comparable things in Windows.

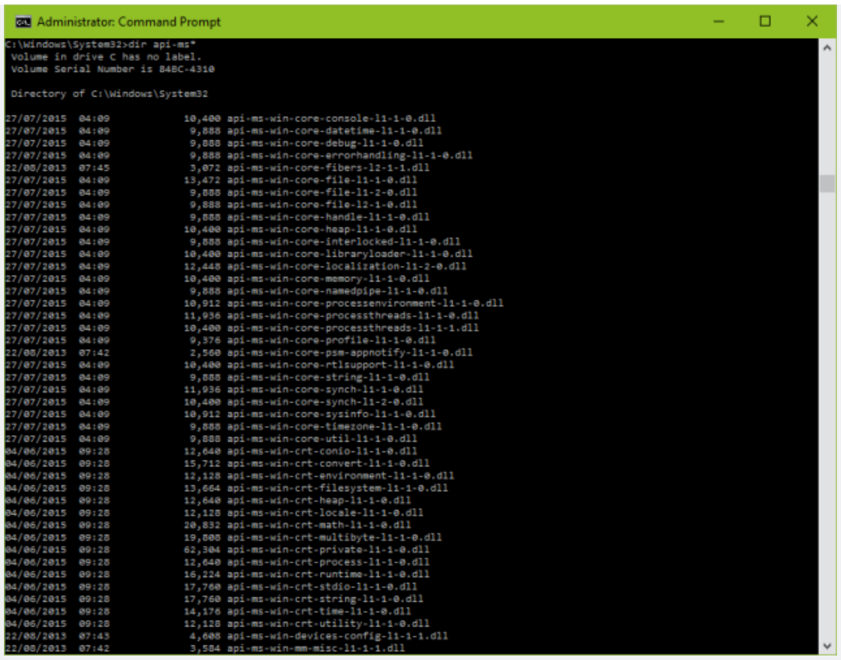
Simultaneously, there was a desire to make Windows development itself more manageable. Windows NT had grown sprawling and complex, with many different interdependencies between components. Microsoft was discovering this made it very hard to predict the impact of changes to the core operating system. Work in one area would unexpectedly have impact on other parts of the operating system, making isolated development impossible.

The plan with MinWin was to repackage and reorganize the operating system to untangle these interdependencies and create an operating system that was much more amenable to rapid innovation without knock-on effects. This work had to be done in such a way as to not break existing applications. The core Windows APIs are organized into DLLs that applications consume. These DLLs can combine features from many different subsystems—kernel32.dll, for example, spans file handling, creating processes and threads, managing memory, loading other libraries, querying certain properties of the system hardware, and many other tasks besides. Microsoft needed to respect and retain this bundling while still enabling separate development.

The interdependencies were particularly acute in areas such as the shell and the browser. These are high-level components, but low-level components often took dependencies on them. Taking these dependencies wasn't elegant, but it was often convenient. Such functionality was useful and also efficient, as it meant reusing the same code and saving on both disk and memory footprint.

The MinWin project painstakingly analyzed the various dependencies within Windows and broke the operating system down into a bunch of components. These were strictly layered to ensure that low-level components never depended on higher-level ones. Dependencies had to strictly flow from high level to low. That way, high-level parts could be safely modified without breaking lower-level parts and without those changes rippling to the broader operating system.

The APIs themselves were also broken down and divided into much more focused units. Portions of this are visible if you go poking around the Windows directories; there are lots of DLLs with names starting `api-ms-win` and `ext-ms-win`, with each of these DLLs listing the functions that govern a particular piece of functionality.

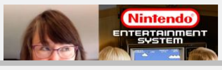


```
C:\Windows\System32>dir api-ms*
 Volume in drive C has no label.
 Volume Serial Number is 84BC-4310

 Directory of C:\Windows\System32

27/07/2015  04:09            10,400 api-ms-win-core-console-l1-1-0.dll
27/07/2015  04:09             9,888 api-ms-win-core-datetime-l1-1-0.dll
27/07/2015  04:09             9,888 api-ms-win-core-debug-l1-1-0.dll
27/07/2015  04:09             9,888 api-ms-win-core-errorhandling-l1-1-0.dll
22/08/2013  07:45             3,072 api-ms-win-core-fibers-l2-1-1.dll
27/07/2015  04:09            13,472 api-ms-win-core-file-l1-1-0.dll
27/07/2015  04:09             9,888 api-ms-win-core-file-l1-2-0.dll
27/07/2015  04:09             9,888 api-ms-win-core-file-l2-1-0.dll
27/07/2015  04:09             9,888 api-ms-win-core-handle-l1-1-0.dll
27/07/2015  04:09            10,400 api-ms-win-core-heap-l1-1-0.dll
27/07/2015  04:09             9,888 api-ms-win-core-interlocked-l1-1-0.dll
27/07/2015  04:09            10,400 api-ms-win-core-libraryloader-l1-1-0.dll
27/07/2015  04:09            12,448 api-ms-win-core-localization-l1-2-0.dll
27/07/2015  04:09            10,400 api-ms-win-core-memory-l1-1-0.dll
27/07/2015  04:09             9,888 api-ms-win-core-namedpipe-l1-1-0.dll
27/07/2015  04:09            10,912 api-ms-win-core-processenvironment-l1-1-0.dll
27/07/2015  04:09            11,936 api-ms-win-core-processthreads-l1-1-0.dll
27/07/2015  04:09            10,400 api-ms-win-core-processthreads-l1-1-1.dll
27/07/2015  04:09             9,376 api-ms-win-core-profile-l1-1-0.dll
22/08/2013  07:42             2,560 api-ms-win-core-psm-appnotify-l1-1-0.dll
27/07/2015  04:09            10,400 api-ms-win-core-rtlsupport-l1-1-0.dll
27/07/2015  04:09             9,888 api-ms-win-core-string-l1-1-0.dll
27/07/2015  04:09            11,936 api-ms-win-core-synch-l1-1-0.dll
27/07/2015  04:09            10,400 api-ms-win-core-synch-l1-2-0.dll
27/07/2015  04:09            10,912 api-ms-win-core-sysinfo-l1-1-0.dll
27/07/2015  04:09             9,888 api-ms-win-core-timezone-l1-1-0.dll
27/07/2015  04:09             9,888 api-ms-win-core-util-l1-1-0.dll
04/06/2015  09:28            12,640 api-ms-win-crt-conio-l1-1-0.dll
04/06/2015  09:28            15,712 api-ms-win-crt-convert-l1-1-0.dll
04/06/2015  09:28            12,128 api-ms-win-crt-environment-l1-1-0.dll
04/06/2015  09:28            13,664 api-ms-win-crt-filesystem-l1-1-0.dll
04/06/2015  09:28            12,640 api-ms-win-crt-heap-l1-1-0.dll
04/06/2015  09:28            12,128 api-ms-win-crt-locale-l1-1-0.dll
04/06/2015  09:28            20,832 api-ms-win-crt-math-l1-1-0.dll
04/06/2015  09:28            19,808 api-ms-win-crt-multibyte-l1-1-0.dll
04/06/2015  09:28            62,304 api-ms-win-crt-private-l1-1-0.dll
04/06/2015  09:28            12,640 api-ms-win-crt-process-l1-1-0.dll
04/06/2015  09:28            16,224 api-ms-win-crt-runtime-l1-1-0.dll
04/06/2015  09:28            17,760 api-ms-win-crt-stdio-l1-1-0.dll
04/06/2015  09:28            17,760 api-ms-win-crt-string-l1-1-0.dll
04/06/2015  09:28            14,176 api-ms-win-crt-time-l1-1-0.dll
04/06/2015  09:28            12,128 api-ms-win-crt-utility-l1-1-0.dll
22/08/2013  07:43             4,608 api-ms-win-devices-config-l1-1-1.dll
22/08/2013  07:42             3,584 api-ms-win-mm-misc-l1-1-1.dll
```

Enlarge / One of the few manifestations of MinWin is all these weirdly named DLLs.

The MinWin name is also used, more specifically, to refer to a minimal kernel, networking stack, storage, and other low-level components. Add in Windows update, the security and authentication infrastructure, device handling, and a few more pieces, and you have CoreSystem.

Page: 1 2 3 Next →



# Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

## Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

## Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

## Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

## API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

**LAW FIRMS**
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

**FINANCIAL INSTITUTIONS**
Litigation and bankruptcy checks for companies and debtors.

**E-DISCOVERY AND LEGAL VENDORS**
Sync your system to PACER to automate legal marketing.

**WHAT WILL YOU BUILD?** | **sales@docketalarm.com** | **1-866-77-FASTCASE**

fastcase® Smarter legal research.