Assessing the Windows 8 Development Platform

Introduction

At the Build conference in September 2011, Microsoft provided details about their next operating system release, code name "Windows 8." Leading up to this conference there has been a fair amount of uncertainty about the future direction of the Microsoft development platform, including Microsoft .NET and Silverlight.

Microsoft revealed that Windows 8 supports two broad categories of application: traditional desktop applications and the new WinRT, or "Metro" style applications.

The new WinRT API and the Metro style applications it enables may represent the future of smart client development on the Windows operating system. However, it is important to understand that Microsoft stated their clear intent that all applications that run today on Windows 7 will run in the Windows 8 desktop environment. This means that applications built using Silverlight, WPF, Windows Forms, or other existing technologies will continue to run on Windows 8.

WinRT is a new operating system programming interface (API), updated for modern technologies and concepts. It replaces the aging Win32 API, enabling the creation of applications that can better take advantage of modern networking, power, and user experience technologies.

"Metro" is a set of user experience and interaction design guidelines that Microsoft recommends for WinRT applications. Microsoft describes Metro as a language for touch-based applications. To this end, Metro defines a language for touch gestures comparable to the existing language for mouse gestures, along with a set of UI style, animation, and interaction guidelines for applications.

No current technologies directly map to the WinRT/Metro environment. The current technology that is closest to WinRT is Silverlight. Developers using Silverlight today will find that their skills remain relevant, and that much of their code can be rewritten for WinRT with reasonable effort.

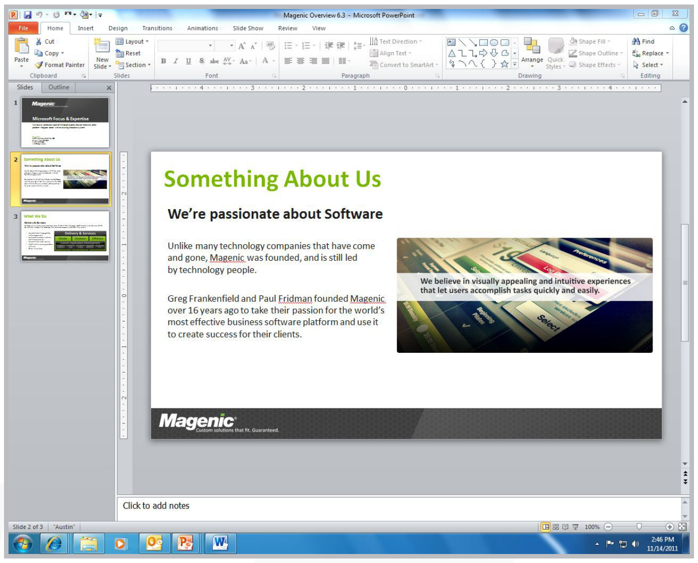
Developers using WPF and HTML5 today will find that their skills transfer to WinRT, but it is unlikely that existing WPF or web application code will easily move to WinRT.

This document will explain the Windows 8 development environment, based on the current understanding of the technology. It will then discuss high level migration strategies from today's technologies to WinRT.

Metro Style vs. Traditional Windows Applications

WinRT/Metro style applications differ from the traditional "Windows" look by eliminating the Windows "chrome" such as frames, window borders, control corners, etc. in favor a full screen, immersive experience. Metro style applications are intended to leverage asynchronous features in the UI controls and languages to provide a very "fast and fluid" interface.

Figure 1 is an example of a traditional Windows application.



Something About Us

We're passionate about Software

Unlike many technology companies that have come and gone, Magenic was founded, and is still led by technology people.

Greg Frankenfield and Paul Fridman founded Magenic over 16 years ago to take their passion for the world's most effective business software platform and use it to create success for their clients.

We believe in visually appealing and intuitive experiences that let users accomplish tasks quickly and easily.

Click to add notes

Figure 1. Traditional Windows application

Notice the following UI elements:

- Title bar with control corners
- Ribbon/toolbar with many controls
- Visual scrollbar controls
- Status bar with information
- Busy visual appearance
- Window borders even when full screen

Figure 2 shows a typical Metro style application. Notice the following differences:

- The page is full screen.
- The page is "chrome" free.
- Visual display is not cluttered, and is easy to read and understand.

Magenic®
Custom solutions that fit. Guaranteed.

- There are no large scroll bars. Instead, there are visual clues that there is more to the right (a well-written Metro style application only scrolls in the "natural" direction – in this case left and right).
- The entire experience is "touch ready," but works with a keyboard and mouse equally well (e.g. touch is a first class citizen). Designing for touch will support mouse and keyboard in most cases.

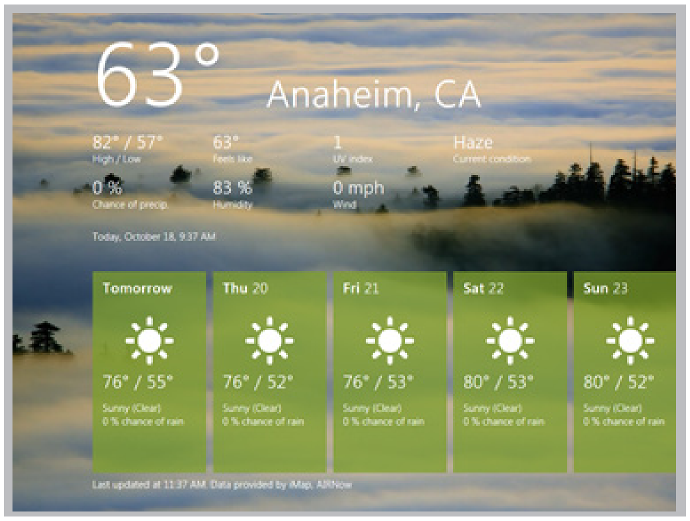


Figure 2. WinRT/Metro style application

WinRT vs. Win32

The Windows Runtime API (WinRT) replaces the old Win32 libraries for accessing operating system functions. The WinRT API is object oriented, largely asynchronous, and callable easily by a wide variety of programming languages.

Win32 continues to be supported by Windows 8, and it is what enables all existing Windows 7 applications to function in the Windows 8 desktop environment.

Figure 3 illustrates the Windows 8 development platform, showing the WinRT and Win32 APIs, along with the development technologies supported by each API.

The Win32 API continues to support existing technologies, including:

- Silverlight
- WPF
- Windows Forms
- C++ (MFC and ATL)
- Browser-based applications using HTML, JavaScript, ActiveX, Flash, and Silverlight

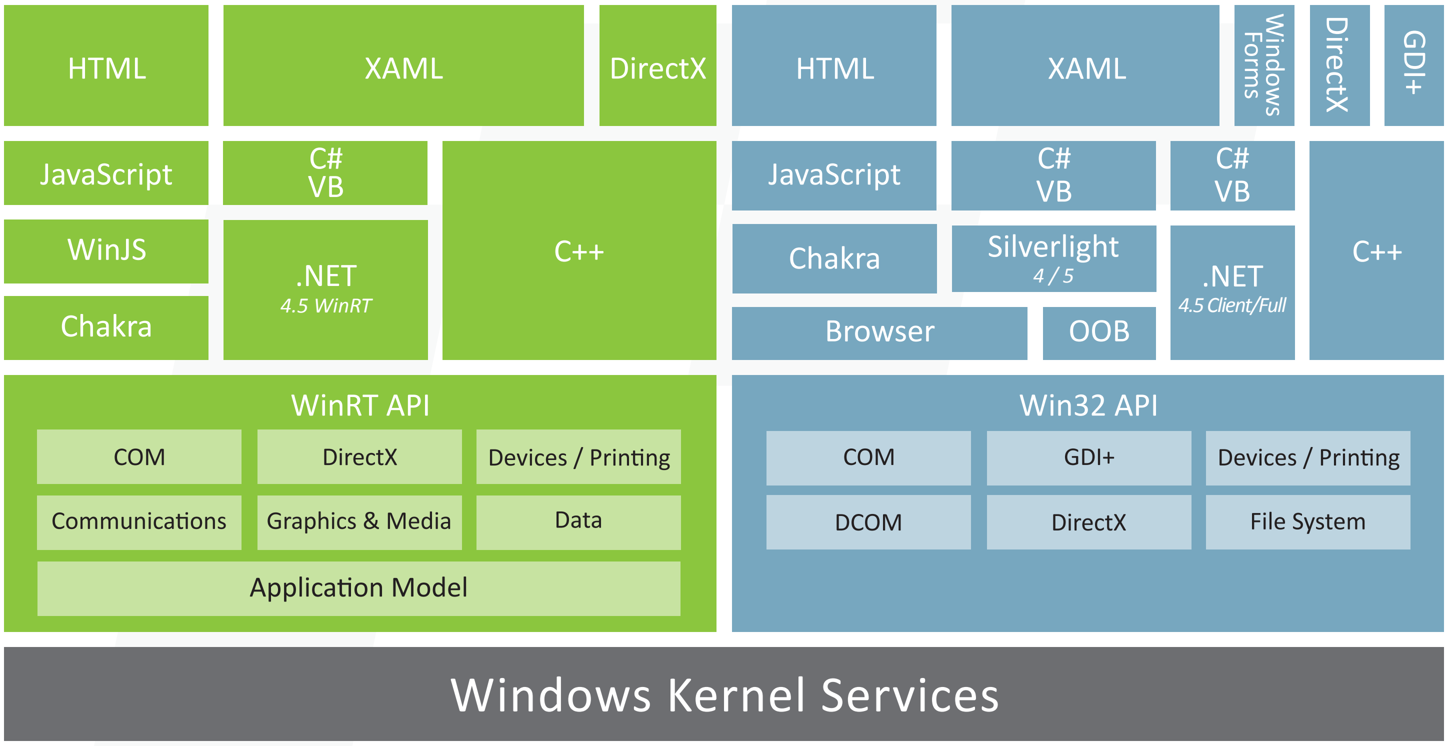


Figure 3. Windows 8 development platform.

The new WinRT API supports the following technologies:

- .NET
- C++
- HTML5 and JavaScript
- Browser-based applications using HTML and JavaScript

The .NET, C++, and HTML5 application models are restricted to the WinRT API and functionality allowed within the WinRT security sandbox. The browser that runs in WinRT does not allow plug-ins, so custom toolbars, Flash, and Silverlight are all off limits.

The main advantages to the WinRT API are as follows:

- Sandboxed security model with restricted functionality that is deemed safe within the sandbox
- Simpler and more stable API as compared to the older Win32 API (improved memory management and stability)
- Support for an easy, asynchronous, object-oriented programming model
- Callable by all supported development tools and languages
- Easy access to hardware such as the camera, sensors, and other modern hardware devices in few lines of code

In summary, Windows 8 offers two broad application development models: WinRT and Win32. The Win32 API allows existing applications to run on Windows 8. The WinRT API enables the creation of new applications that can take advantage of modern hardware, networking, and other services provided by the new API.

Windows 8 Development Strategy

When considering the impact of Windows 8 on future software development, the following broad strategies should be evaluated:

1. Continue to use existing technologies, and run the application in the desktop environment.
2. Create a WinRT/Metro style smart client application that takes full advantage of the new WinRT and Windows 8 features.
3. Create a browser-based web application that relies on no plug-ins, so it can run in the browser in both the WinRT and desktop environments.

The first two options are the most likely options if your current applications are smart client applications that use WPF, Silverlight, or Windows Forms.

Although it is possible to write a new web application to replace existing smart client applications, this involves a completely different developer skill set, and offers no ability to reuse any existing code assets. However, it is also the case that web applications provide the broadest reach of all application types. Such applications can run on any device with a browser, including all major platforms and devices, such as Windows, Mac, iPhone, iPad, Android, etc.

The third option is ideal if your current applications are web applications, because it is likely that your existing web applications will continue to offer the same behaviors and value to the end user in Windows 8 as they do today.

In summary, you must first decide whether to create smart client applications or web applications. If you decide to create smart client applications, you must then decide whether to support WinRT, or to build applications for the Win32 desktop environment.

The remainder of this paper assumes you are considering the creation of smart client applications on WinRT.

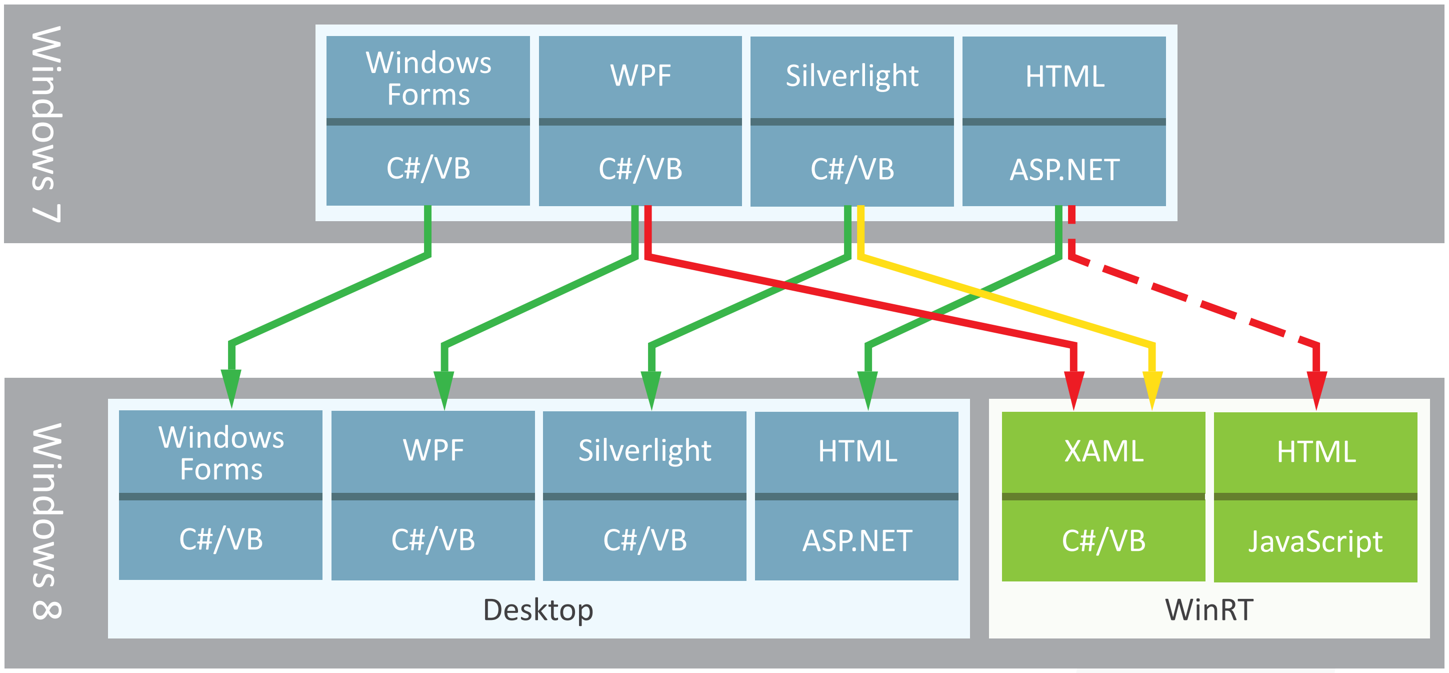


Figure 4. Mapping current technologies to Windows 8.

Conversion Strategies

No existing technologies map directly to the WinRT platform. Figure 4 shows how existing technologies map to the Windows 8 development platform.

As you can see, all existing technologies map directly to the Windows 8 desktop environment. This is illustrated by the green lines, indicating that these applications are expected to work in Windows 8 with no effort.

The yellow line for Silverlight indicates that many Silverlight applications can be migrated to WinRT with reasonable effort. We will discuss this in more detail later in the paper.

The red line for WPF indicates that migration to WinRT is possible, but will require more substantial effort.

The red dashed line for HTML indicates that development skills will transfer, and a limited amount of existing HTML, CSS, and code assets may apply to WinRT application development.

Applications written using existing technologies will require effort to migrate to WinRT. For applications written with technologies other than Silverlight and WPF, the term "rewrite" is probably more accurate than "migrate".

Common Migration Scenarios

Silverlight to WinRT .NET

Silverlight provides a reasonable migration path to WinRT. We come to this conclusion because of the following:

- Silverlight and WinRT use XAML to describe the UI layout and interaction.
- The WinRT subset of .NET is not that different from the existing Silverlight subset of .NET.
- Silverlight and WinRT both require asynchronous interaction with servers, so Silverlight applications are already architected and designed to be asynchronous.

It is important to understand that Silverlight applications won't "just run" on WinRT. Although they use similar XAML and have a similar .NET subset, there are enough differences that any migration effort will require some reworking of the XAML and application code. We expect substantial XAML and code asset reuse, but with some effort.

One important consideration is to use clear layering and separation of concerns when building Silverlight applications. Applications should avoid all "code-behind" the XAML controls, and should use the MVVM (model-view-viewmodel) design pattern to cleanly separate all code from the XAML.

WPF to WinRT .NET

WPF shares some common technologies with WinRT (and Silverlight). These include the XAML language for describing the UI layout and interaction, along with the C# or VB languages, and the .NET base class library.

However, WPF is less likely to provide an easy migration to WinRT than Silverlight. We come to this conclusion because of the following:

- WPF provides access to the full .NET API, and it is quite likely that existing WPF applications make use of .NET features that don't exist in WinRT.
- Few WPF applications use asynchronous interaction with servers, and moving synchronous code to an asynchronous model usually requires a lot of effort.

Notice that these assumptions can be overcome. If you are extremely careful to apply some constraints to your use of WPF, the migration process can be more reasonable. Specifically:

- Avoid using parts of the .NET API that aren't available in Silverlight (such as direct file system access).
- Use only asynchronous server access.
- Maintain clear layering and separation of concerns.
- Avoid all "code-behind" the XAML controls, and use the MVVM design pattern.

By following these guidelines when building WPF applications today, the chances of reusing existing XAML and code assets are increased.

Web Applications to WinRT HTML5

Web applications share some common technologies and concepts with HTML5 smart client applications in WinRT, but there are fundamental differences. Web applications assume the use of a web server, and the majority of web application code typically runs on the web server. An HTML5 smart client application doesn't have a web server, and all application code runs on the client.

Existing web development skills in HTML and CSS layout, along with JavaScript programming knowledge, will apply to HTML5 WinRT development. Optimistically, it is possible that some HTML and CSS markup can be moved from a web application to a WinRT HTML5 application, as can any JavaScript code that is highly focused on user interaction and HTML manipulation.

The high level of architectural difference means that all server-side code will have to be rewritten into JavaScript in the smart client application, or refactored into service interfaces that can be installed on an application server to be called from the application.

In summary, existing applications will not run in WinRT without change. Of all current technologies, Silverlight offers the best potential for migrating code assets. Existing WPF and HTML skills will carry over to WinRT development, but little or no existing code assets are likely to carry into the new platform.

Conclusion

The new WinRT API and Metro style applications it enables may represent the future of smart client development on the Windows operating system. Existing applications will continue to run in the Windows 8 desktop environment. Additionally, existing web applications that avoid the use of plug-ins will run in the WinRT web browser.

If you decide that the WinRT and Metro style application model is a platform you may wish to support in the future, your best strategic move is to start developing today using Silverlight. Alternately, you can use WPF with extreme care to emulate the Silverlight development model.

In any case, existing developer skills in XAML, C#, VB, .NET, and Silverlight carry forward to WinRT development. The same is true for HTML5, CSS, and JavaScript developer skills.

Contributing authors: Kevin Ford, Jason Bock, Sergey Barskiy, Stuart Williams, Chris Williams and Rockford Lhotka.

Assessing the Windows 8 Development Platform

Appendix A: Technology Comparison Chart

Application Environment	U/I Technology to Use									
	Windows Forms	WPF	Silverlight / Moonlight	WinRT .NET	WinRT Other	Web Forms [4]	ASP MVC	Mobile Enabled Web Apps	Mono for Android / iOS	Native WP7 / Android / iOS
OS Requirements										
Supports full Windows environment, multi versions	Yes	Yes	Yes	No	No	Yes	Yes	No	No	No
Supports full Windows 8 only environment	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No
Supports MacOS/Linux	No	No	Yes	No	No	Yes	Yes	No	No	No
Supports other OSes	No	No	No	No	No	Yes	Yes	No	No	No
Supports mobile	No	No	No	Yes [2]	Yes [2]	No	No	Yes	Yes	Yes
Application Requirements										
Supports touch	No	Yes [1]	Yes [1]	Yes	Yes	No	No	Yes	Yes	Yes
Supports access to Windows API/3D	No	Yes	No	Yes	Yes	No	No	No	No	No
Creates native rich client apps	Yes	Yes	Yes	Yes	Yes	No	No	No	Yes	Yes
General Requirements										
Supports app store retail distribution	No	No	No	Yes	Yes	No	No	No	Yes [6]	Yes
Needs enterprise client app deployment (e.g. SCCM)	Yes	Yes	No	Yes [5]	Yes [5]	No	No	No	Yes [5]	Yes [5]
Leverages strong internal .NET skillset	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	No [7]
Leverages strong internal HTML/JS Skillset	No	No	No	No	Yes [3]	Yes	Yes	Yes	No	No

* Note: combinations of factors not listed here may point to two or more UI technologies being needed.

1. If touch is needed in the future, application can be designed to Metro style standards.
2. Windows 8 phones are planned to be supported with Metro applications.
3. HTML5/JS Metro applications could be selected if there is an internal skill set for that technology.
4. WebForms are generally considered an older technology, in general use ASP MVC unless there is a compelling reason to use WebForms.
5. For applications not in the app store, a deployment system will be needed.
6. Mono may make app store approval more difficult.
7. Windows Phone 7 leverages .NET skill sets.

Magenic®
Custom solutions that fit. Guaranteed.