Porting Android* Apps to Windows 8* - Overview

Objective

Developers looking to port their existing Android apps to the new Windows 8 Platform face several challenges. In this article we will provide an overview of the differences between the two platforms, and show how developers can go about porting their apps. We will compare developer tools in both platforms and provide an overview of app framework differences. We will discuss basic app creation, packaging, and app store differences. Fundamental app concepts like UI and input handling will be discussed. This article will serve as a foundation for other articles that delve into more advanced topics in porting Android apps to Windows 8.
Windows 8 is a major revision for one of the predominant desktop OSs in the industry. From a developer point of view, it brings major changes to the way Windows applications are developed, designed, and built. It is redesigned and reimagined from the ground up to support new user experiences and to take advantage of all the capabilities of modern mobile devices like tablets and UltraBook™ devices.

With the introduction of new form factor devices like smartphones and tablets, more and more software is being targeted for these platforms. Currently there are hundreds of thousands of apps in Android and other mobile ecosystem stores.

Developers who already have an existing Android app might want to consider targeting the new Windows 8 OS. These two OSs, their apps, and development environments, one might argue, are radically different. But at a higher level they also have some common features like support for touch, sensors, mobility, and other hardware features. There are lot of benefits and opportunities in developing for Windows 8. In this article, we will give an overview of differences and cover the basics of porting Android apps to Windows 8.

We can broadly classify the porting challenges into development environment and tools, app framework differences, and the overall OS/ecosystem differences.

Below are few basic snapshots of how Android and Windows 8 OS look and differ:
Figure 1: Android® Home Screen and “List of Apps” screen (captured from Android Emulator)
As seen in these two screenshots, the two OSs differ dramatically. The new Windows 8 UI removes all non-app related notifications from the main screen (like the notification bar at the top and other icons). You will notice new Windows 8 apps focus on content and remove the chrome of traditional app UI as much as possible. Also, Windows 8 introduced a unique UI feature called tiles (or live tiles) where each app on the home screen in Windows 8 is represented as a box or a window resembling tiles. The content of these tiles can be dynamically updated by the owning app with app-relevant data or information, and are hence referred to as “Live” tiles.

We will go into other details shortly, but these are some of the basic and important UI differences.

**Development Environment and Tools**

When considering porting an app from one platform to another, one of the important areas developers will encounter is developer tools and programming environment. The current industry trend in this area is to provide an integrated development experience as much as possible. Companies try to provide tools for all stages of development—coding, debugging, testing, building, packing the apps, and even uploading the apps to stores.

Developers coming from the Android platform and looking to port their apps to Windows 8 will find that the programming environment and development tools in Windows 8 are comprehensive and powerful.
In Windows 8, developers can choose their programming language of preference. Multiple programming languages are supported—C++, C#, Visual Basic*, and JavaScript*/HTML5. All of these programming languages are first-class citizens, meaning they all have access to full system capabilities and use the same common Windows 8 API (Windows Runtime).

Another important area is the integrated development environment (IDE). While Android developers use Eclipse* with Android plugin (ADT), Windows 8 developers use Visual Studio* 2012, which has been updated to support the new Windows 8 platform. Blend for Visual Studio 2012 will assist in Window 8 app UI (XAML based) interactive design and testing. Blend also supports HTML5 apps UI design and CSS3 styling. These tools can be downloaded from the website below:


Use Visual Studio 2012 with Windows 8 for Windows Store app development, as the new apps depend on the Windows 8 platform and APIs.

Similar to default project templates for apps available in Android, Visual Studio 2012 comes with templates for different kinds of Windows 8 apps and programming languages.

The screenshots below show the templates for Javascript and C++:

![Figure 3: JavaScript* Project Templates (captured from Visual Studio* 2012)]
Figure 4: Visual C++ Project Templates (captured from Visual Studio® 2012)

Visual C# and Visual Basic have similar project templates too.

Android developers wanting to fully rewrite their apps for Windows 8 might want to consider using one of the above project templates as a baseline for their apps.

Windows 8 has a powerful UI design tool—Blend for Visual Studio 2012. It is fully integrated into the Visual Studio IDE and makes the development life cycle as simple as possible. Blend can be directly invoked from inside the VS 2012 IDE. Blend has support for both XAML-based UI design as well as the HTML5/CSS3 apps.

Below is a screenshot of Blend used in designing a C#/XAML app:
Blend is a very powerful and comprehensive UI design tool. It supports interactive UI editing/design and quick testing. For example, in Blend for HTML5/CSS3 apps, we can edit the HTML DOM and CSS3 styling live, and immediately see the result. This is very powerful for trying out different styling properties, layouts, or themes.

Below is a screenshot of Blend used in designing a HTML5/CSS3 app:
Blend for Visual Studio 2012 has powerful testing tools for different display sizes and views.

The next screenshot illustrates how developers can test their Windows 8 apps for different views and screen sizes supported by the Windows 8 platform:
Another developer tool that is common in modern app development is the emulator or simulator. In Android we have the Android Emulator with support for different configurations through AVD. Windows 8 has a powerful simulator that is fully integrated with Visual Studio 2012 IDE and app development life cycle. It simulates several of the common platform features like touch, geo-location, camera, device orientation, and different form factors and screen sizes.

Windows 8 simulator can be invoked directly from VS 2012 IDE. The screenshot below shows simulator capabilities and features:
We also need a way to test our app on a real device. In Android we use the ADB and its Eclipse ADT integration to remotely test our apps on devices. Windows 8 apps can be tested on remote devices using the Remote Debugging Tools for Visual Studio. We will need to install the remote tools on the target device, and configure them before we can use it with Visual Studio remote debugging. This web site provides a comprehensive guide to running Windows Store apps on a remote machine:


In Visual Studio, we can choose our target of choice for debugging: Simulator, Local Machine, or Remote Machine. The next screenshot shows these Debug options in Visual Studio 2012:
Another important development environment topic is that of app building, packaging, and app stores. In Android, Eclipse with ADT plugin integrates and automates the process of building, signing, and packaging the apps. Windows 8 provides comprehensive support for the entire app development life cycle from developing, building, signing/certificates, packaging, and direct integration with the Windows Store.

The table below provides a summary of the different topics we discussed in this section (please note it is not comprehensive so may not have all the details).

<table>
<thead>
<tr>
<th>Feature</th>
<th>Android*</th>
<th>Windows 8*</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDE</td>
<td>Eclipse* with ADT Plugin</td>
<td>Visual Studio* 2012, Blend for VS 2012</td>
</tr>
<tr>
<td>Programming Environment</td>
<td>Java*, C/C++ with JNI/NDK, HTML5 inside Android Webview, OpenGL*</td>
<td>C++, C#, Visual Basic*, XAML, JavaScript*/HTML5, DirectX*</td>
</tr>
<tr>
<td>SDK</td>
<td>Android SDK, NDK, Google APIs, Android SDK manager tool</td>
<td>Windows 8 Runtime APIs, Windows Services APIs such as Bing SDK, etc.</td>
</tr>
<tr>
<td>Debugger</td>
<td>Java Debugger (JDWP), ADB, DDMS, Traceview, GDB</td>
<td>Visual Studio Debugging Tools. Depends on programming language and APIs – JavaScript debugging, Managed Code debugging, DirectX debugging, etc.</td>
</tr>
<tr>
<td>Device Emulator</td>
<td>Android Emulator, AVD</td>
<td>Windows 8 Simulator</td>
</tr>
<tr>
<td>Device Access &amp; remote debug</td>
<td>Android ADB, DDMS</td>
<td>Visual Studio Remote Debugger Tools</td>
</tr>
<tr>
<td>Testing</td>
<td>Use different emulator configurations with AVD, Eclipse Android Tools for UI design, software unit testing</td>
<td>Windows 8 Simulator, Expression Blend* for quick interactive UI design and testing, software unit testing</td>
</tr>
<tr>
<td>App Packaging</td>
<td>APK archive file with the app, manifest, resources, and other files</td>
<td>APPX archive file with the app, package manifest, resources, and other files</td>
</tr>
<tr>
<td>Store</td>
<td>Google Play and other 3rd party stores</td>
<td>Windows Store</td>
</tr>
</tbody>
</table>
To port apps from Android to Windows 8, we need to study the differences between these two platforms. These two platforms have totally different application stacks—from the low level app interactions with the OS, to the highest abstractions like the UI.

Before we delve into the details, we need to note one important caveat—Windows 8 supports two types of applications: the new Windows 8 apps (called “Windows Store apps”) and the traditional Windows applications (“Desktop apps”).

Existing desktop applications continue to work and behave the same on Windows 8. However, now existing desktop applications can take advantage of new features like Touch, Sensors and other new Windows 8 platform features. But some of the new Windows 8 features like live tiles, the new app life cycle, security sandbox, etc., might not be available in desktop apps.

The web page below provides an overview of the new Windows Store apps:


At a higher level, Android apps can be composed of three important pieces:
- Application components (Activities, Services, Content Providers, Broadcast Receivers)
- Manifest file (defines user permissions, declares app components used, app capabilities, requirements, and so on)
- Application resources (images, media, language strings, locale support, resources for different device types, etc).

Windows Store apps also have similar pieces, though they may not map exactly to Android pieces:
- Different application components like the main app component, app contracts, extensions, background tasks, notifications/tiles, etc.
- Package manifest for the app (defines package identity, properties, capabilities/permissions, extensions, visual elements, etc.).
- App resources and localization (Package Resource Index PRI file, all app resources are indexed and associated)

The screenshot below shows the capabilities and available declarations in the Windows 8 Package Manifest of a sample app:
Both Android and Windows Store apps have a well-defined app life cycle, or different states the app goes through from start to end. The link below provides an overview of the app life cycle of Windows Store apps:


Similar to Android, Windows 8 support different views for the app—portrait mode, landscape mode, different screen sizes, etc. In addition to these, Windows 8 supports two more views not available on Android – called “Snap View” and “Filled View”.

The screenshot below shows a Windows 8 app in four different views – landscape, portrait, snapped view (bottom left, Windows store app is in filled view), filled view (bottom right, Windows Store app is in snapped view).
Figure 11: Sample Windows 8* App in different views (captured from Windows 8 Simulator)

For guidelines on how to design for snapped and fill views, please refer to:

Another important area of difference is the common app controls and UI surfaces. In Android we have the concept of an “Action Bar” for common app actions or buttons and other common app features like settings (app preferences), search integration, and so on. In Windows 8 the closest equivalent concept would be “app bar” and “Charms.”

The app bar is the primary command interface for Windows Store apps. Similar to the “Action Bar” in Android, we can use it for navigation and other app actions. By default the app bar is hidden in Windows 8 apps, it can be invoked by a special Windows 8 gesture. Its behavior can be customized as needed (e.g., always stay active, dismiss on touch, etc.). The next screenshot shows an app bar in action in Windows 8 Internet Explorer* app:
The app bar can be added either at the bottom, the top, or both. It can be invoked by a swipe gesture either from the top or bottom of the screen. With keyboard-mouse input, it can be invoked by right clicking inside the app.

Charms are a unique feature in Windows 8 apps. They are a specific, common set of buttons always shown in every app when charms are invoked (by a special Windows 8 gesture). The buttons are: Search, Share, Start button, Connect to Devices, and Settings.

The next screenshot shows the Charms bar, invoked inside a Windows Store app:
The Charms bar is usually positioned on the right-hand side of the screen and invoked by a swipe gesture from the right side of the device.

Another area of difference is obviously the programming languages and platform APIs. Since Android apps use the Java programming language, to port the app to Windows 8 the app has to be rewritten in one of the programming languages supported by Windows 8. Depending on the app features, requirements, and capabilities, there might be equivalent platform APIs available in Windows 8 that developers can take advantage of. For example, the touch and sensor functionality can be ported by using the equivalent Windows 8 APIs. Please visit this site for a comprehensive API reference for Windows Store apps:


In this section we provided an overview of different areas of porting challenges that Android developers face when converting their apps to the Windows 8 platform. As we have seen, many high level app concepts overlap. Developers can map their app design and logic to corresponding or equivalent patterns in Windows 8. Sometimes the conversion might not exactly map because that feature is not available or it is implemented in a totally different pattern. In these cases, a rewrite of that particular feature/component needs to be considered. Further, the apps can be enhanced to take advantage of the unique features of Windows 8 like live tiles, new kinds of app views, etc. This will help provide the best user experience for your app on Windows 8.

**App UI and Input Handling**

The previous section provided overall conceptual differences between Android apps and Windows 8 apps. In this section, we will build on that and cover how to port one of the first and basic pieces of any app: UI and Input Handling.

In Windows 8, an app’s UI can be built in a few different ways:

- Using XAML with C++/C#/.VB
- Using HTML5/CSS3 with JavaScript
- Direct graphics interface using DirectX/C++ (some XAML can be mixed here)
- Desktop apps UI

Developers should carefully consider different aspects of the target app when picking one of these choices. It depends on app requirements and planning for future extensibility and reuse. For example, if the existing app uses lots of web-based UI content or already has significant HTML code, developers should seriously consider developing the app in JavaScript/HTML5. On the other hand, if the app requires heavy graphics and full GPU control, DirectX/C++ might be considered. As noted previously, all programming languages have the same access to all the system capabilities (via common Windows 8 Runtime API).

Apps that require desktop features, with several UI controls and chrome, may consider using the Desktop app model.
Once we pick our programming environment of choice, we can proceed to look at other UI aspects like window management, views handling, event handling, styling, UI controls, animations, etc. These differ quite a bit depending on the underlying environment. For example, HTML5 apps have their own event handling patterns (as defined by HTML DOM standards and web practices), and styling capabilities (CSS3).

In Android apps, the UI of a typical app consists of multiple “Activity” components, which host “View” and “GroupView” objects that implement the UI elements like buttons and layouts. The UI life cycle, back stack, input and event handling all are taken care of by this framework. Windows 8 has a very powerful UI framework and programming model, which has common system-wide UI concepts:

- Common design style – typography, layouts, navigation, animations
- Touch interaction – well-defined, system-wide touch gestures, with new features like semantic zoom
- Snapping and scaling – support for new views and different screen layouts and sizes
- Common UI controls – e.g., buttons, list view, app bar, charms, settings, etc.
- Tiles and notifications

A comprehensive discussion about UI for Windows Store apps can be found at link below:

It is strongly recommended for developers to take advantage of Blend for Visual Studio 2012 to design UI for Windows 8 apps. Blend supports both XAML-based UI design, as well as HTML5/CSS3 UI design. It has comprehensive tooling and design support, and allows for interactive and quick UI verification/testing.

Developers will also need to make design choices for how to best take advantage of all the UI features and capabilities offered by Windows 8. The article below titled “Design case study: iPad* to Windows Store app” covers some of the UI design choices we need to consider when porting apps from other platforms:

For a full step-by-step case study of how to port the UI and touch input, from Android to Windows 8, please refer article “Porting Android Touch to New Windows* 8 UI(C#)” on Intel Developer Zone(IDZ).

**Summary**

Porting apps from Android to Windows 8 takes a lot of planning and effort. The porting process can be simplified and more efficient, if we fully understand how Android platform's features and capabilities compare to the ones available in Windows 8. If developers conceptually map their app logic and design to equivalent features in Windows 8, they will be able to achieve the best possible port.
In this article we provided an overview of differences between the two platforms, and discussed how Android platform features compare or translate to Windows 8. We covered the basics like development tools, programming environment, platform and framework differences, and UI concepts. We hope this article will serve as a foundation for future articles that delve into more advanced and in-depth topics.

**Notices**

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked “reserved” or “undefined.” Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to:
http://www.intel.com/design/literature.htm

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations, and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Any software source code reprinted in this document is furnished under a software license and may only be used or copied in accordance with the terms of that license.