# Strategies for App Communication between Windows* 8 UI and Windows 8 Desktop


## Abstract

Windows 8 WinRT API allows developers to create and deploy apps quickly, and to publish those apps in the Windows Store. When an app needs access to lower-level system resources, the Windows 8 Desktop APIs are needed. To get both together, developers must create two apps, one for each environment, with some method of communication between them. For apps sold in the Windows Store, this communication cannot be done locally. As stated in the Windows Store certification requirements:

> **3.1b** Windows Store apps must not communicate with local desktop applications or services via local mechanisms, including via files and registry keys.

This article discusses the main approaches for communication between Windows 8 UI apps and Windows 8 Desktop apps, including the design considerations in deciding which to use, and the basics of implementing each approach. If a network connection is not an option, the required local communication would prohibit the apps from being listed in the Windows Store. With intranet connectivity, viable options include web services and a shared cloud. Expanding to the internet allows large-scale commercial solutions for shared storage and other web services.


## Overview

Windows 8 UI apps are intended to be sleek, reliable, fast, and touch-oriented. Window 8 UI apps are restricted in what parts of the file system, OS, and hardware they can access. Updating existing apps to this model introduces obstacles, with some app functionality being impossible to implement in the WinRT API. One solution is to create a Windows 8 UI front-end that communicates with a Window 8 Desktop app to perform the work not allowed by the WinRT API. There are a few ways to do this, limited by the Windows Store requirements.

Businesses developing their own Windows 8 UI apps for internal use do not need to distribute apps through the Windows Store. As a result, these apps are not subject to the local sharing restriction for Windows Store apps. More info is available at http://blogs.msdn.com/b/windowsstore/archive/2012/04/25/deploying-metro-style-apps-to-businesses.aspx.

## Considerations

### Direction of Communication

If your lightweight app is simply serving up data for user consumption, communication will be directed primarily (if not entirely) from the Desktop app to the Windows 8 UI app. It's a rare case for the situation to be reversed, but not entirely unthinkable. If the front end is a more interactive user interface, it will need to communicate back and forth with the back end.

### App Switching or Background Communication

If the communication required is discrete, like saving a file in a lightweight editor and switching to a fully-featured editor, the communication method can be more static. This affords more options in terms of implementation. Continuous communication brings more restrictions. Among the biggest is that this communication must originate from the Windows 8 UI app running as the front end. If the roles are reversed, with the Desktop app running in the foreground, its Windows 8 UI counterpart is likely suspended. A suspended app is essentially frozen in state, unable to communicate or process information.

### Connectivity

Both apps being on the same machine limits the options drastically because many inter-process communications available in previous systems are not available to connect WinRT with Win32-based applications. Increasing the scope to an intranet adds a few more options. A local cloud or server could store the files, notifications can be pushed using the Windows Push Notification Service, or web services can be used. If the machines have access to the internet, large-scale cloud storage becomes viable, as does using external web services.

### Standalone

For a Windows 8 UI app to be deployed publically, it must first be uploaded to the Windows Store, which requires that it pass certification. Windows 8 Desktop apps can also be listed in the store, but must be standalone (not requiring another piece of software be installed). Windows 8 UI apps in the store can depend on other software, but only if that software is also listed in the store. While Windows Store apps can make use of other programs (such as a server providing content) this can be a large obstacle to developing apps that work together. The Desktop app needs to have at least solid basic functionality, with the Windows 8 UI app serving as a companion program (adding more functionality) or an enhancement (improving the functionality that already exists). If the Desktop app is the back end, supplying the entire content of the Windows 8 UI app, it needs its own front end to serve its purpose independently.

## Viable Approaches

### Web Service

A Windows 8 Desktop app can be running as a backend with web service exposure, allowing the Windows 8 UI app to connect and communicate. The Desktop app can be also used as a mediator, handling the interactions and receiving requests over the connection. You can see an example of how to use web services from the Windows 8 side at http://www.codeproject.com/Tips/482925/Accessing-Data-through-Web-Service-in-Windows-8-St.

### Local Files

When the two apps are on the same machine and are attempting to communicate without using the network, the limited options make things difficult but not impossible. As long as the deployment is internal and does not need to use the Windows Store, the apps could communicate via local files they can both access. If the Desktop UI app avoids some write-lock pitfalls, it can modify the shared files to enable communication in almost real time by writing and reading the files quickly and often. The Intel

Energy Checker SDK uses a similar model for instrumentation (more information available in References). Since named pipes, shared memory and other standard inter-process communication methods are disabled in Windows 8, using local shared files is the main remaining option for this approach. The danger involved is that of user exposure; since the files are in a shared folder, users can access, view, and modify the files. This provides some security issues if the files are unencrypted and functionality issues if the user decides to lock, change or delete the files. This approach is detailed at http://stackoverflow.com/questions/7465517/how-can-a-metro-app-in-windows-8-communicate-with-a-backend-desktop-app-on-the-s.

### Cloud - Storage and Notifications

If network access is available, both of the apps can share files on a remote server or data cloud. Most cloud services have safeguards in place to prevent file access collisions and data loss. Setting up a Windows Push Notification Server (WNS) allows the Desktop app to send messages and updates as notifications to the Windows 8 UI app. The notifications can be "toast" style, live tile updates, or handled by code for customized communication. You can see an example of how to use toast notifications at http://code.msdn.microsoft.com/windowsapps/Toast-notifications-sample-52eeba29 and information on a cloud backend at http://www.windowsazure.com/en-us/develop/mobile/tutorials/get-started.

### Simulated Style

If the above options cannot be used (e.g. a one-machine setup with sensitive communication), one solution is to drop the Windows 8 UI app entirely, instead using only a launching/shortcut tile to a fullscreen Desktop designed in the sleek Windows 8 UI style. While not gaining access to all the Windows 8 features, this allows full development without WinRT restriction. You can find information and examples of this type of app at http://stackoverflow.com/questions/12881521/desktop-application-in-metro-style.

## Dead Ends

### Local Web Connection Loopback

A web connection loopback can be used by applications on other operating systems to connect via a web socket to another program on the same machine. There are problems with this approach on Windows 8. Windows 8 has security features in place to disable a loopback. This security can be disabled manually, but it will only work in debug mode. Aside from making it prohibitive for commercial deployment, the Windows 8 UI app would never pass certification for listing in the Windows Store.

### Win32 Communication

Windows 8 permits Windows 8 Desktop APIs to be packaged in a managed DLL for use by Windows 8 UI apps. Although this sounds promising in principle, it brings a host of new problems to the table. The DLL would need to be packaged with the app, bloating the size. If an app needs some Win32 functionality that does not violate the Windows Store certification policy, a custom library could be the right solution. While adding APIs to WinRT may be a good idea in specific cases, it's not viable as a general strategy for inter-app communication.

## Summary of Viable Approaches

| Approach | Requires Network | Exposed to User | Two-Way Communication | Other restrictions |
|---|---|---|---|---|
| Local Files | No | Yes | Not entirely, Windows 8 UI app must be the front end | Must be side-loaded, cannot use Windows Store |
| Cloud / Windows Notification Server (WNS) | Intranet+Server | No | Yes for cloud, but WNS is directional toward Windows 8 UI only | Many cloud services require internet access or in-network server host |
| Web Services | Intranet+Server | No | Yes | Many web service options require internet access or in-network server host |
| Style Simulation | No | No | N/A | No live tiles, can't use charms from Windows 8 Desktop, possible user confusion |

## Conclusion

There are many ways to get the benefits of both the Windows 8 UI and Desktop applications. As with any software solution, there isn't a universally best option. Various restrictions limit the selection, but there are benefits to each method. When Internet access is readily available, cloud and web services are the most widely applicable approach. The information presented here is a starting point to help you make the right decision for your apps, and more options may become available in the future.

## Other References


Windows 8 App Certification Requirements http://msdn.microsoft.com/en-us/library/windows/apps/hh694083.aspx

Windows 8 Desktop App Certification Requirements http://msdn.microsoft.com/en-us/library/windows/desktop/hh749939.aspx

Windows Server App Certification Requirements http://msdn.microsoft.com/en-us/library/windows/desktop/hh848078(v=vs.85).aspx


## About the Author


Brad Hill is a Software Engineer at Intel in the Developer Relations Division. Brad investigates new technologies on Intel hardware and shares the best methods with software developers via the Intel Developer Zone and at developer conferences. He is currently pursuing a Master of Science degree in Computer Science at Arizona State University.