**Pre-requisites**

* [Install Google Chrome](https://www.google.com/chrome/) (needed for Postman)
* [Install Postman](https://www.getpostman.com/)
* Install Visual Studio 2015 Update 3
* Go to Windows **Settings** **Update & security** **For developers**  select **Developer mode**
* [Install **Desktop App Converter** from the Store](https://www.microsoft.com/en-us/store/p/desktopappconverter/9nblggh4skzw)
* [Download the **BaseImage-14393.wim** file](https://aka.ms/converterimages)
  + Guide will assume this is saved in **C:\BaseImage\**
* [Download the demo files](https://microsoft.sharepoint.com/teams/dtms/vstech/myhealth/Shared%20Documents/Forms/AllItems.aspx?id=%2Fteams%2Fdtms%2Fvstech%2Fmyhealth%2FShared%20Documents%2FMyHealth%5FIgnite%2FDemo%5FFiles%2FDemo%5FDesktop) 
  + Guide will assume these are saved in **C:\IgniteDemos\**
  + Extract Certs.zip into **C:\IgniteDemos\Certs\**

**Postman setup**

* Run **Postman**
* Click on **Import** (top-left) and select **C:\IgniteDemos\WNS.postman\_collection.json**

**Desktop App Converter (DAC) setup**

* Run **Desktop App Converter** as Administrator
* Run the following command to allow for execution of the DAC
  + Set-ExecutionPolicy bypass
* Run the following command to set up the base image for the containers (if the Containers feature isn’t enabled, a restart will be necessary)
  + DesktopAppConverter.exe -Setup -BaseImage C:\BaseImage\BaseImage-14393.wim

**Certificate setup**

* Run **Developer Command Prompt for VS2015**
* Run the following command to add the code signing certificate to the trusted certificate store
  + Certutil -addstore root “C:\IgniteDemos\Certs\DesktopBridgeIgnite.cer”

**Demo 1 - Converting an app using the Desktop App Converter (4 minutes)**

*Key points:*

* *Desktop App Converter is one way to do the conversion: manual packaging is available for apps that don’t have an installer, InstallShield, WiX and Advanced Installer support outputting a Windows app package out of the box*
* *Goal of conversion is to simplify installation for the developer, optimize the installation, update and uninstallation process for consumers (cleaner, easier, no chain installers)*
* *After conversion, app package can be distributed through the Windows Store or existing distribution channels*

**Demo:**

**Inform the audience of pre-work (30 seconds)**

* Open the **Windows Store app** on the machine and do a search for “Desktop Bridge”
* Click on the **Desktop App Converter** and show that is has been installed
* Navigate to **C:\IgniteDemos\** and show that the Open Live Writer setup has been downloaded (**OpenLiveWriterSetup.exe**)

**Convert Open Live Writer installation (2.5 minutes)**

* Run **Desktop App Converter** as Administrator
* Paste in the following command for the conversion
  + DesktopAppConverter.exe -Installer C:\IgniteDemos\OpenLiveWriterSetup.exe -InstallerArguments "/S" -Destination C:\IgniteDemos -PackageName "OpenLiveWriter" -Publisher "CN=8534DD2F-8D54-4DEF-A235-21DEF9658A9A" -Version 0.6.0.0 -MakeAppx -Verbose -AppExecutable openlivewriter.exe -PackageArch x86
* Call out the following in the command:
  + -InstallerArguments “/S” is being used to run the installer silently, this is a requirement for the Desktop App Converter, as it spins up a container inside Windows and captures the changes (no user interaction possible)
  + -PackageName and -Publisher are inserted in the app’s new package manifest
  + -AppExecutable is specified if there are more executables created during installation, so the entry point of the app package can be specified in the manifest
* Run the command to start the conversion
* After conversion, navigate to **C:\IgnitesDemos\OpenLiveWriter\** to show the output:
  + **OpenLiveWriter.appx** file is the app package that has been created
  + **logs** folder contains the DAC logs in case you need them
  + **PackageFiles** is the extracted app package contents, which can be used to replace the default Assets and tweak the app manifest for example
* Show double-click on **OpenLiveWriter.appx** displays the installation dialog
  + Note that app name is missing spaces, publisher display name is just the GUID and the tile image is the default one, these can be changed in the manifest
* Clicking on **Install** prompts an error message that the package isn’t signed: all app packages must be signed with a trusted certificate (the Windows Store does this for you when you distribute through it)

**Sign the converted OLW app package (1 minute)**

* Run **Developer Command Prompt for VS2015**
* Run the following command to sign the app package
  + signtool.exe sign -f C:\IgniteDemos\Certs\DesktopBridgeIgnite.pfx -fd SHA256 -v C:\IgniteDemos\OpenLiveWriter\OpenLiveWriter.appx
* Explain that this is a self-signed certificate that we have already added to the trusted root certificate store
* Once signed, double-click the **OpenLiveWriter.appx** package again and complete the installation
* Launch the app to show that it starts the familiar OLW first-run experience

**Demo 2 - Adding UWP APIs (11 minutes)**

**Walk through the app and code (6 minutes)**

* In the Demo 2 folder, show the installation experience of the Healthclinic.biz.appx and launch the app to show what the starting app looks like
  + The app will fetch appointments from the cloud service and update the list in the app
  + We're going to add a Live Tile to the app and show the push notification APIs
  + This is only possible with a converted app, because it requires an app identity, which is only available for apps in a Windows app package
* Open up the MyHealth.Client.Desktop solution
  + Show the basic structure of the app, with the Portable Class Library to share common logic across platforms and the WPF project for Windows desktops
* Go to the **App Data cache** bookmark and explain that this is how you would write this code, but that the App Data folder gets redirected for converted apps (in this example, to the **C:\Users\[user]\AppData\Local\Packages\477Rajen.Healthclinic.bizIgnite2016\_jaqeghtgaj9rj\LocalCache** folder)
* Open the **References** of the project and highlight the **Windows** reference. This metadata file holds all the UWP APIs and is located in **%programfiles(x86)%\Windows Kits\10\UnionMetadata**
  + There are separate metadata files located in **%programfiles(x86)%\Windows Kits\10\References**
* Go to the **Push notifications** bookmark and explain that this is the UWP API to get a push notification channel for the current user and app, which can be used to send notifications to from the web
  + Note that a couple lines down in this example we're copying the channel to the clipboard and writing it to a temp file in App Data, but normally you would store this channel with a reference to the user in a cloud service/database
* Go to the **Live Tiles** bookmark and explain this is the UWP API to update the main tile of the app on the Start menu
  + In this case, we're grabbing the XML of one of the default templates (TileSquare150x150Text01) and fill it with the data for the next appointment
  + We then create the actual notification from the XML and send the notification to the update manager, which in turn updates the Live Tile

**Register the app and show Live Tile functionality (2 minutes)**

* Now, because we have added these UWP APIs, we can't simply F5 to debug, because the APIs need the package identity (you can show the exception that occurs if you do). In VS15 (next version), we'll have a project to enable this debugging experience from inside Visual Studio. For now, we'll build the app and we'll manually register the app to easily run/debug it.
  + Note that this requires the PC to be in **Developer mode**, which you probably enable anyway as it's needed to develop UWP apps.
* Build the app and open a PowerShell window
* In the PowerShell window, use the following command to register the app directly in Windows
  + **Add-AppxPackage -Register "[…]\Demo 2\MyHealth.Client.Desktop\bin\Debug\AppxManifest.xml"**
* Show that the app has appeared in the Start menu list of applications and that Pin it to Start
* Run the app and let it update the list of appointments. After it has updated, show that the Live Tile was updated with the next appointment information

**Show push notification functionality (3 minutes)**

* Run **Postman** to send the push notification
  + Note that the channel URI, as needed below, is automatically copied to clipboard when running the app
* Go to **Collections** **WNS** in Postman, select **1. Access Token** and hit **Send**
  + This is requesting the access token from WNS to get permission to send push notifications to users of the app. A package SID and client secret are required, but already pre-configured in this case
* Create a new tab in Postman and select **2. Toast notification** from the WNS collection
  + This sends the actual push notification to the device
* Paste the URL in the clipboard into the URL field next to **POST**
  + If it’s no longer in the clipboard, just launch the Healthclinic.biz app again
* Go to the **1. Access Token** tab and copy the “access\_token” from the response body
* Go to the **2. Toast notification** tab and paste the copied access token into the “Authorization” header field, but keeping the “Bearer<space>” in front of it
* Press **Send** to send the push notification to the device
* Bonus: close the app and send the push notification again to show that it arrives even when the app isn’t running

**Demo 3 - Adding a UWP component (8 minutes)**

**Walk through the solution (6 minutes)**

* Show the three different projects in the solution
  + BackgroundTasks - background task for toast notifications
  + MyDesktopApp - the WPF desktop app
  + MyUWPApp - the UWP app
* Show MyBackgroundTask.cs and highlight the APIs to do toast notifications
* Show Program.cs in MyDesktopApp
  + Show the registry APIs in Main()
  + Show the app service registration in Main() and ThreadProc()
  + Show Connection\_RequestReceived()
    - Receiving ValueSet (Message)
* Show App.xaml.cs in MyUWPApp
  + Show the app service registration in OnBackgroundActivated()
* Show MainPage.xaml
  + Show the UWP page with a fancy ColorAnimation
* Show MainPage.xaml.cs
  + Show registration of the background task registration in OnNavigatedTo and RegisterBackgroundTask
  + Explain that MainPage\_StatusUpdated is registered as the callback for the app service message that gets sent back
  + Show LaunchBackgroundProcess and how it launches the full trust process (explained in a moment in the manifest)
    - Note that this requires the Desktop extensions SDK to be referenced, show the Reference
  + Show comboBox1\_SelectionChanged and how it sends to message to the app service
    - It also updates the Live Tile
* Show the Package.appxmanifest (make sure you show the code, not the visual designer), couple highlights:
  + rescap and desktop XML namespaces declared
  + TargetDeviceFamily on Windows.Desktop (remember, this only works on PCs)
  + Extensions for background task and app service
  + desktop:Extension for fullTrustProcess
    - Highlight that this has to be in the AppX folder of the app package to be able to use the full trust process launch API
  + rescap:Capability for runFullTrust
    - This is a restricted capability, that will need special permissions to go to the Store (process explained earlier)
* Show the MyUWPApp project Properties
  + Show the Build tab
    - Compile with .NET Native tool chain enables, so .NET Core will be used
  + Show the Build Events tab
    - As mentioned, the package should have the full trust process in the AppX folder
    - Also, the MyUWPApp build output is moved to a folder, so we can easily register it as an app
* Because the build takes a few seconds using .NET Native tool chain, we'll use the build that's already there

**Register and run the application (2 minutes)**

* Open PowerShell and enter the following command to register the app
  + Add-AppxPackage -Register "[…]Demo 3\MyUWPApp\bin\x86\Debug\AppxManifest.xml"
* Find **MyDesktopAppStep4** in the Start app list and Pin the app to Start
* Run MyDesktopAppStep4 and change the status, showing the Live Tile updates
* In a minute the toast notification should show up as well

**Demo 4 - Getting started (8 minutes)**

* Open the Edge browser and show the following sites:
  + <http://www.flexerasoftware.com/installshield> - InstallShield website to get information and the product
  + <https://developer.microsoft.com/windows/bridges/desktop> - Main Desktop Bridge site that has the latest information, documentation and videos
    - Highlight the feedback and help links at the bottom (feedback on UserVoice, help in the forums)
  + <https://developer.microsoft.com/windows/projects/campaigns/desktop-bridge> - Submission form to let us know you are ready to submit the app to the Store
    - This kicks off a review process and approval for submitting with the runFullTrust capability in the Windows Store
  + <https://msdn.microsoft.com/windows/uwp/porting/desktop-to-uwp-root> - Main docs site
    - Prepare your desktop app for conversion to UWP - shows some of the things you need to know (highlight a few)
    - Start the conversion process - shows the different options to convert your app/use the bridge
  + <https://msdn.microsoft.com/en-us/windows/uwp/porting/desktop-to-uwp-extensions> - extensions added in UWP for apps using the bridge, examples:
    - Startup tasks to declare a startup task that runs when the user logs on (for example, to put an icon in the system tray. An API is available (on the page) to enable/disable this during runtime
    - App execution alias - alias for cmd/Run dialog to run the application. This no longer requires messing with the user's PATH, this declares it with Windows directly
    - Main point here is that we have a bunch of ways to extend into the OS, but no longer needs registry/persistent changes on the machine, so it stays clean
  + <https://msdn.microsoft.com/en-us/windows/uwp/porting/desktop-to-uwp-deploy-and-debug> - Deploy and debug information
    - Support is coming in VS15 to create a project in the solution to deploy/debug immediately, without having to register the app or attach the debugger
    - Deployment information is right below it, such as the Add-AppxPackage -Register command we've been using in this session