Developers Guide to Windows 10 – Anniversary Edition Demos

# Setup:

1. Collect IP address for the Pi3, and the Windows Phone
2. Enable the device Portal for both devices and launch them in the browser in its own tab. (Sometime Edge doesn’t like the urls and does a Bing search. I’ve found using Win + R for the URL works great)
   1. [On Xbox, open Dev Home and enable the device portal with username and password. Open the portal in your browser tab (it will be https://ip:11443)]
   2. For Windows Phone it’s in the developer settings. Make sure to disable Authentication. The IP address will be on this screen, navigate to that ip in the browser
   3. Login to the device portal for the Pi 3 (<http://ip:8080> user/pass Administrator/p@ssw0rd) NOTE: http, not https
3. Cortana demo loaded in VS ‘15’
4. Pen ready
5. Uninstall Win32App

# Demo 1 – XAML updates

## step-by-step

1. XAML Edit and Continue:
   1. Start the application and at the Login Screen then do a Side-By-Side with Visual Studio
   2. I’m now going to show you a neat feature that everyone has been asking for and we’ve finally got it coming.
   3. Now go towards the bottom of **MainPage.xaml** where you will see an image that can be uncommented out. Also change the margin a little too.
   4. Yes, that’s right we now have Edit and Continue in XAML, this great feature will save us so much time. Long gone are the days where we will be stopping the debugger, making a small change and then relaunching the app.
   5. Those that are also eagled eyed have also noticed that I also added an animated gif to the screen too. We now support animated gifs, I’m expecting so many apps to now ship with cat pictures!
2. Access Keys
   1. Now onto my next feature. You know as developers we hate using the mouse, we are keyboard wizards and I know we are not alone. Well now in the Anniversary update we will be supporting Access Keys and it’s really easy to implement.
   2. Open up **Dashboard.xaml**
   3. Find the buttons in the **Relative** **Panel** where you will see the **AccessKey** defined in the XAML. These are the keyboard shortcuts that will drive the keyboard experience when used with the Alt key.
   4. However, we also want to highlight the tooltip to show what the access keys are when the user just pressed the alt key. Similar to what you see in Office. To do this I am going to wire up 2 events.
      1. **AccessKeyDisplayRequested**
      2. **AccessKeyDisplayDismissed**
   5. Open **Dashboard.xaml.cs** and look at the SetupTooltip, Menu\_AccessKeyDisplayRequested and MenuButton\_AccessKeyDisplayDismissed.
   6. You can see that all I am doing is creating the Tooltips needed for each access key and placing them to the right of the control. Just simple additions to your code can make the experience so great.
3. ItemsUpdatingScrollMode:
   1. Ok so next is a feature which I for those of you who write apps that have chat features will love. It’s the ability to keep adding items to your listview and have it scroll to the last item in the list. Let me show you.
   2. Make sure that app is launched and on the main menu select **Messages**. Start typing some words into the chat window so that it starts scrolling.
   3. So let’s take a quick look at the XAML for this.
   4. Open **Message.xaml**, you can see in the ListView that all I’ve had to do is change the ItemPanelTemplate with an ItemStackPanel setting the ItemsUpdatingScrollMode to KeepLastItem. That’s it!
4. XAML Binding:
   1. Ok so I’ve shown you quite a few things UI related, but about those folks in the audience who deal with complex data heavy apps. Well I have something for you too.
   2. Make sure the app is launched and select Report on the menu, then click all the items in the sub menu showing titles, a number and currency.
   3. As you can see from my reports, clearing the meetings I go to aren’t that interesting, especially about (read title of report)
   4. What’s more interesting though is how I was able to bind to this data.
   5. Open up **ReportViewModel.cs** and show the Data Property. Dictionaries are so useful but it’s just annoying when trying to bind to this. Not any more though!
   6. Open up **Report.xaml** and scroll to the bottom.
   7. You will see 2 lines of XAML using some of the new bindings.
   8. The first line shows me binding to a dictionary, you can see I’m looking up the key ‘budget’ in the Data property. The number of times we’ve had to create a complex object type just so we can access something that replicates a dictionary. Isn’t this really cool?
   9. However, we didn’t stop there!
   10. Highlight the call to the method in the binding on the next line!
   11. Look I can now call a method from XAML! You can see that I’m taking the selected item and then calling my C# method.
   12. Open **Report.xaml.cs** to the **GetFormattedText** method
   13. You can see that I take the **ReportViewModel** as the input and return a string.
   14. Just imagine all those times you’ve had to create a converter so you can perform various functions.
5. These are just some of the investments we are making in the XAML platform to make your lives easier.

# Demo 2 – Windows Ink

* File New Project
* Add InkCanvas, with x:Name=”MyInkCanvas”
* Add InkToolbar, HorizontalAlignment=Right, TargetInkcanvas=”{x:Bind MyInkCanvas}”
* Demo multimodal input with ruler

# Demo 3 – More Devices, Less work

* 1. Let's look at IntelligentKioskSample – the UWP test app for Cognitive Services
  2. In VS, Run app to Desktop (x86) to show it – Do the Face recognition demo
  3. Stop debugging.
  4. Since this app targets UWP, can run on all devices that support the universal windows platform. Let's have a look at some of them:
  5. Device Portal:
     1. Universal Windows Devices such as the Xbox, Phone and IoT (and Hololens) provides a simple way to access developer data and settings right from your browser
  6. Phone and IoT:
     1. (switch to **Phone portal**) here it is connected to this Phone here where you can enable the Device portal in the phone Settings – Show developer settings
     2. Enable **Project My Screen** - position side-by-side with Dev Portal - Launch app from Dev Portal OR start from tile on start screen
     3. But that's not all! Switch to **IoT Device Portal tab**. Go to Remoting to check it is enabled
     4. Launch **Windows IoT Remote Client** and connect to the Pi - position side-by-side with Device Portal
     5. Launch the app from IoT Device Portal as well
        1. Do the Emotion demo
  7. So there we have it - one app running on three different devices
     1. IoT: Show Windows IoTRemote Client
     2. Phone: Show Project My Screen
     3. Desktop: launch PhotoBoothPro app
     4. Say would also run on an Xbox if we had one here !

# Demo 4 – Composition

## Demo

We want to show you some great effects in this demo which you can apply to your applications. As we mentioned before Composition is very fast and close to the metal so we are going to demonstrate this using an expensive effect – Gaussian blur. None of what you are seeing here is pre-rendered, it’s all on-the-fly.

## Files needed

1. MainPage.xaml.cs – MainPage\_Loaded

## step-by-step

1. Load Cortana sample app demo (same as in Demo 1) to show what you are going to explain
   1. Point out the blur and opacity when the app loads
2. Stop Debugger and go to MainPage\_Loaded method (MainPage.xaml.cs)
   1. We first grab the compositor, this gives us access to the Visual Layer. We do this by using the helper class ElementCompositionPreview. We convert the page to a Composition visual and then ask for the compositor of that visual.
   2. Create a Gaussian Blur effect
      1. Import to point out
         1. Name is “Blur”
         2. BlurAmount is 0
         3. Source is a parameter called source
   3. We use the effect factory to take the blur effect we created and create a shader, we have asked composition to not bake in the BlurAmount into the shader as we are animating that value. We then create a brush which we can apply to a sprite later.
   4. Composition by default adds easing, we want to ensure we get linear in this case
   5. We create an animation:
      1. using CreateScalarKeyFrameAnimation method.
      2. All animations run between a 0 to 1. We are specifying that at the end of the animation, we want the value to be a 100f.
      3. We have the animation running for 6 seconds
      4. We then apply the animation to the blurbrush, changing the property Blur.BlurAmount (basically taking Name and Property we called out before and animating it from the 0 to 100 in 6 seconds)
      5. Final part is to set the source of this animation, we create a backdropbrush, as that’s how blurs are applied.
      6. We now need to push this to the UI, we create a sprite and set the brush to the blurbrush we’ve just created.
      7. Using the ElementPreviewComposition we push the sprite back into the XAML tree.
   6. Show that we also have an Opacity animation running at the same time
3. Show briefly the Login\_Click code, which uses animations to fade out
4. Show the Composition demos in the UWP Community Toolkit sample application: <https://www.microsoft.com/store/apps/9nblggh4tlcq>

# Composition – Gears

Show **Windows UI Dev Labs** sample app

## Demo

We get that although the blur stuff is cool that not everyone will want this in their application. Remember Composition is about performance, this is really great for even LoB scenarios. I was working at an Investment bank where the trading tiles are constantly flashing red and green to provide stock information. You can imagine that traders have 9 screens with loads of these flashing tiles, something that is very graphically intensive especially when you need millisecond timing. The next demo I want to show you is going to prove that we can run at 60 frames a second with minimal CPU interference.

## Files needed

1. Gears.cs
   1. AddGear\_Click
   2. AddGear
   3. ConfigureGearAnimation
   4. AnimateFast\_Click

## step-by-step

1. Launch application
2. Point out the number in the top right represents frame rate (should be 60)
3. Click the Slow button, to show the main gear is rotating – just a simple animation
4. Add another gear by clicking the Add Gear button next to the Reverse button. Point out that number in the application shows that number of gears. Keep hitting Add gear till you get to about 10. Remind the audience that already most application don’t run 10 simultaneous animations and still get 60 frames a second (unless you are game)
5. Time to up the ante, in the text box type 500, and explain that this is going to add another 500 gears. Hit the Add X Gears button and watch the cogs increase to 510. Open up the task manager to the Details tab. Find Composition.exe and point out the low CPU usage.
6. Try to challenge the system, so click Fast, check Task Manager again, not much change. Act a puzzled.
7. Add another 1500 gears, so you should now have 2010 gears running. Look in task manager, you should be still less than 10% CPU. Now emphasize – THIS IS A UWP APP RUNNING 2010 CONCURRENT ANIMATIONS AND IT’S TAKING HARDLY ANY CPU! Now while not all applications need this sort of performance, if you are an app dev that has demanding applications Composition is for you.
8. Close the application. Let’s take a look at the code.
9. Open Gears.cs
10. When the user clicks Add Gear it fires the **AddGear\_Click** event
    1. We simply create an image and set the height and width
    2. We also set the position of the gear and then add it to the XAML (Container is a Canvas)
    3. Now we do 2 things, Call the **Add Gear** Method and **ConfigureGearAnimation** method. Let’s look at the AddGear method
11. **AddGear** simply grabs the visual we had just added to the container. We set the height and width and anchor point so it rotates from the center. We add this visual to a list called gearVisuals.
12. **ConfigureGearAnimation** is where some interesting magic happens. We create something called an Expression animation.
    1. We use “-previousGear.RotationAngleInDegrees” to set the degrees.
    2. Notice there is no explicit number being set there.
    3. We then define what previousGear is by setting the reference parameter.
    4. We then take the currentGear and start the animation on the property RotationAngleInDegrees using the expression we just created.
13. That’s pretty much the setup when we add a gear. Pretty simple! So what happens when click Animate Fast.
14. We call the AnimateFast\_Click method. We call the EnsureGearMotor(). So as all the gears are using expressions to drive their animation, all I need to do is just rotate the first gear (the red one) and the one connected to it should rotate because of the expression, and this creates an onward chain. You can see in this method I basically create an animation, although I’ve not started it yet.
15. Back in our AnimateFast\_Click method you can see I set a duration for a full rotation and then I start the animation on the first gear and hey presto!! They all start animating! ☺

# Demo 5 – Web Links

App developers love to market to the users via various media like email. One the biggest challenges with this is driving usage to your application, normally users are always taken to the web site first because there is no way for the email to understand if the user has the application installed. However, with Web Link on Windows you can now launch your app directly and if the user doesn’t have the app it gracefully then takes them to the web site it would have done normally.

## Files needed

1. Package.appxmanifest (ContosoCafe project)

## step-by-step

1. Launch the Contoso Café project to show what it looks like, then close it
2. Now open the mail application where you should see an email from Build Tour with a Free second cup of tea. Mouse over the image to show that the url it goes to is a web site. It also has an offer Id which can be passed to your application too.
3. Click the image, you will now be presented with an app selection dialog.
4. As the application has registered for this web domain it is now appearing an option. Select ContosoCafe app and uncheck the Always use this app box and then and select Ok. The application should now launch. Let’s take a look how to do this.
5. Open the Package.appxmanifest, scroll to the Extension section where you can see an extension registered with a category of windows.appUriHandler.
6. The host has been set with the domain of the web site it wants to register for. It’s as easy as that!!

# Demo 6 – Project Centennial

Background tasks sample

- Press Ctrl+Shift+B, or select \*\*Build\*\* \> \*\*Build Solution\*\*.

- Under the BackgroundTasksSample\cs directory you will find the CentennialPackage to deploy during development, run: "**add-appxpackage –register AppxManifest.xml**" from a PowerShell window.

- To run the app, go to the start menu, find the app tile "Win32App", right click, and choose "Pin to taskbar"

- Go to settings and change your time zone, you will receive notifications from the background task! Your live tile will also receive updates.

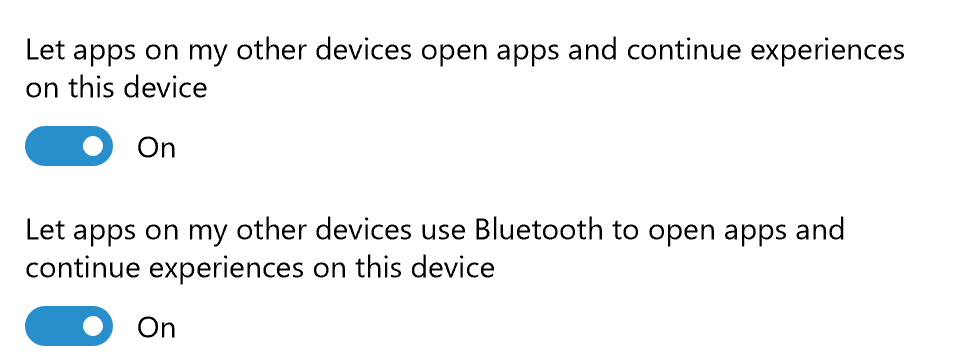
\*NEW\*

1. Create new C# console app
2. Insert snippets
3. Add NuGet UWPDesktop
4. Add new desktop packaging project

# Demo 7 – Project Rome

### Setup

* Deploy standard Rome sample app on two machines – both must be associated with your MSA – set breakpoint at Scenario1.xaml.cs, line 63
* Deploy Roman Test App (<http://aka.ms/romeapp>) on both
* TIP: you are strongly advised to enable Bluetooth on your machines and to enable Remote App Connections in Settings – Privacy:



Much more reliable than using conference WiFi to attempt a cloud connection!

### Demo

* Show display from machine 1 – run the standard Rome sample app
* Stops on breakpoint – step through explaining code.
* Select the other machine from the list of discovered systems – show the LaunchUri scenario screen but before clicking it, show the second machine display to prove that the browser isn’t running (Tip: you can try to show both machines at the same time by remote desktop to your second machine and then dock the Remote Desktop UI to one side of your primary display)
* On the first machine, click the LaunchUri – browser launches on the second machine. Explain that the LaunchUryAsync() api has been extended to work on RemoteLauncher objects (used to describe a remote system discovered in your proximity)
* On the first machone, start the Roman Test App, connect to the second machine and draw on the inking area – show the display of the second machine to demonstrate that the app gets launched remotely on that machine and then echoes the ink strokes you enter on the proimary machine.