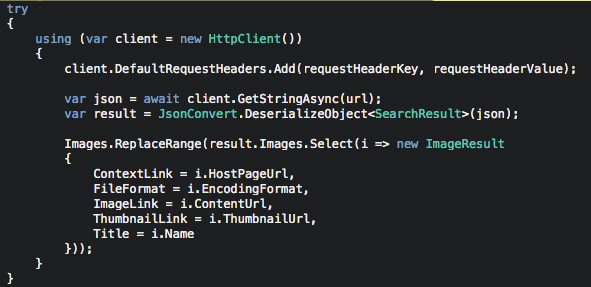
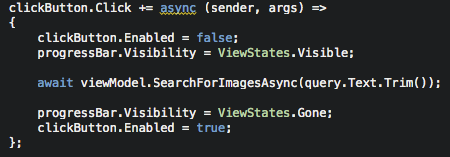
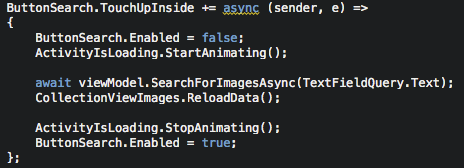
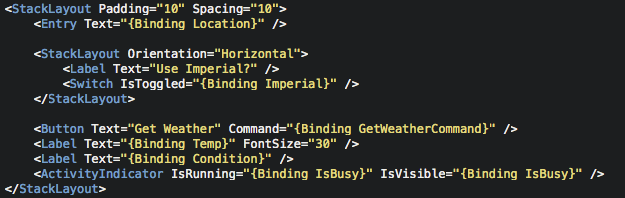
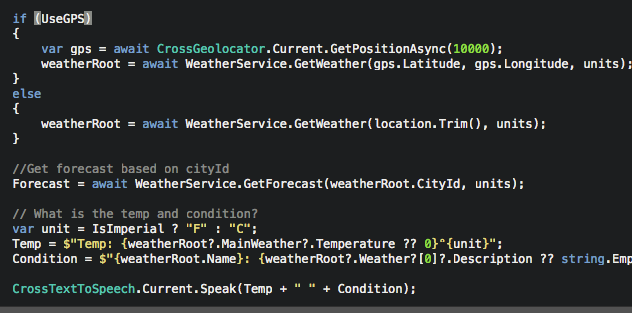
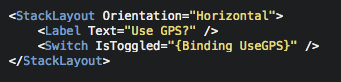
**Demo Checklist**

* Open Blank Visual Studio.
* Open Visual Studio emulator.
* Test deploy a Blank iOS Application to the remoted Simulator.
* Open up app-imagesearch-cogs-start in Visual Studio.
* Test deploy Android Visual Studio emulators.
* Test deploy UWP.
* Set zoom in code to 150.
* Close all windows.
* Open designers for iOS / Android

**Demo 1 - Image Search**

* Build an application powered by Bing Image Search.
* How do we get started?
  + File->New Project
    - Show templates
      * iOS
      * Android
      * Cross-Platform
* Walk through project.
  + Portable Class Library
    - Look at properties and frameworks.
    - What’s in the PCL?
      * Services to actually call into Bing Image Search.
        + Could do many things here - call to your web service or cloud backend.
        + We have some associated models, but all we really care about is ImageResult.
      * View Models / MVVM
        + View models are abstractions of our view.
        + Ideally all non-platform-specific view logic would reside here.
  + Android
    - Using activities / layouts
  + iOS
    - Using view controllers / storyboards
* Let’s add some functionality to this baby!
  + View model
    - Add code to actually search for our image.
    - Let’s use some awesome C# features and pull down some images from Bing Image Search.
      * Using (var client = new HttpClient())
      * client.DefaultRequestHeaders.Add(reqeustHeaderKey, requestHeaderValue);
    - We need to do network calls and long-running on background threads.
      * Explain async/await/Tasks.
      * Var json = await client.GetStringAsync(url);
    - Now we have a JSON string.
      * Need to deseralize this into an object using JSON.NET.
        + Using Components/NuGets - great thing about .NET.
      * Var result = JsonConvert.DeserializeObject<SearchResult>(json);
    - We don’t need SearchResults though, we need ImageResults.
    - We can use LINQ to get around this.
    - 
* Look at Android project references
  + Android support libraries
  + Awesome .NET libraries alongside them
  + Using shared code as well
* Let’s build out a page.
* Open main.axml (already have this open)
  + Opens Android designer.
  + Show toolbox
  + Drag and drop a control
  + Show controls
  + Tap source and see .xml
* MainActivity is starting point for our activity
  + Set main layout for page
  + Use adapter to tell our recycler view (which is like a list view) how each cell is supposed to work
* Let’s add an event!
  + 
* Build & Debug
  + Add breakpoint
  + Search cute monkey
  + Hit breakpoint
  + Search for something else
* Everything so far can be done 100% on a PC, if I want to do iOS development I can still write all my shared code, but if I want to design/deploy, I need to be connected to Mac, no matter where it is.
* iOS app
  + Same as Android
    - References, resources
  + AppDelegate is our entry point of our app
  + Show storyboard = UI and flow of our navigation
    - Show the different form factors, toolbox, etc.
  + Look at code behind - view controller
  + Add click event.
    - 
  + Build & Debug
    - Show iPhone Simulators
    - Deploy!

**Demo 2 - MyWeather**

* Build a Xamarin.Forms app for tracking the weather.
  + Finding location from text or location from device
* Open up app-myweather.
  + Set Android as the startup.
* To get started with Xamarin.Forms, File->New Cross-Platform Forms.
* Go over solution
  + We have all of our shared code here, but what’s new is we also have our UI here (views folder).
  + Xamarin.Forms is distributed via NuGet - so it can be updated easily at any time.
  + We still have iOS, Android, and Windows 10.
    - Explain Xamarin.Forms.
* Open Weather model.
  + Looks similar to Bing models. Have typical weather stuff here.
* Open Weather service.
  + Uses HttpClient to call out to OpenWeatherMap APIs.
  + We can pass lat, long, or city.
  + We can also get a five day forecast.
* Open WeatherViewModel.
  + Implements INotifyPropertyChanged.
    - What is this?
      * Anytime you update this value, notify anybody who’s subscribed to it.
    - Why does it matter?
      * Bindings infrastructure
        + Updates our UI when this value changes.
        + Flow of data between view model and view
  + IsBusy
    - Tells us if we’re already doing work on the background, so we don’t do it again
  + Commanding
    - Flow of events from view to view model.
  + Look at execute weather command method.
    - We go out and get the weather by city, and the forecast.
    - Try, catch, finally
* Let’s build our UI. Open up WeatherView.
  + ContentPage / StackLayout
    - Padding, Spacing
  + Let’s add our UI
  + 
* Forecast View
  + Already done
  + Explain code
* App.cs
  + Entry point for Xamarin.Forms applications
  + We are created a tabbed page
  + Add weather view model as the binding context (this is where the binding/commanding looks).
  + We can manage application lifecycle.
* Run on Android/iOS side by side.
* What if we want to use geolocation or speak back the weather?
  + Manage NuGet Packages
    - Look at Geolocator / Settings / TextToSpeech/
    - What are Plugins?
* Jump back to view model.
  + 
* Let’s update our view
  + 
* Run on iOS
* Run on Android
* Run on UWP