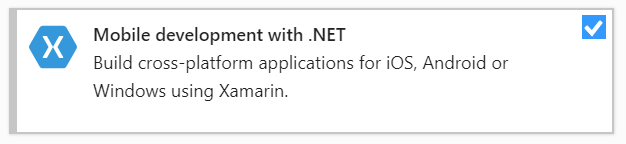
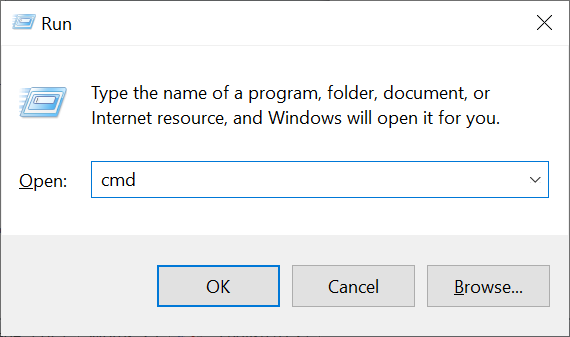
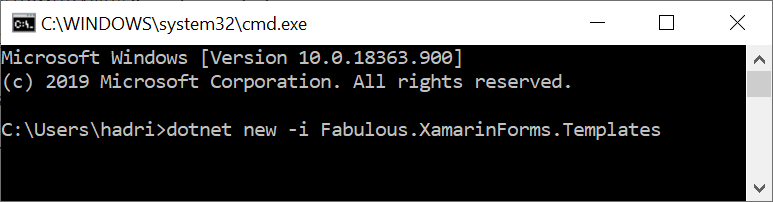
Installation:

* On a Windows computer, download Visual Studio Community <https://visualstudio.microsoft.com/vs/>
* Select “Mobile development with .NET” in the installer
* (For iOS users)
  + Install XCode on your Mac

Downloading the template:

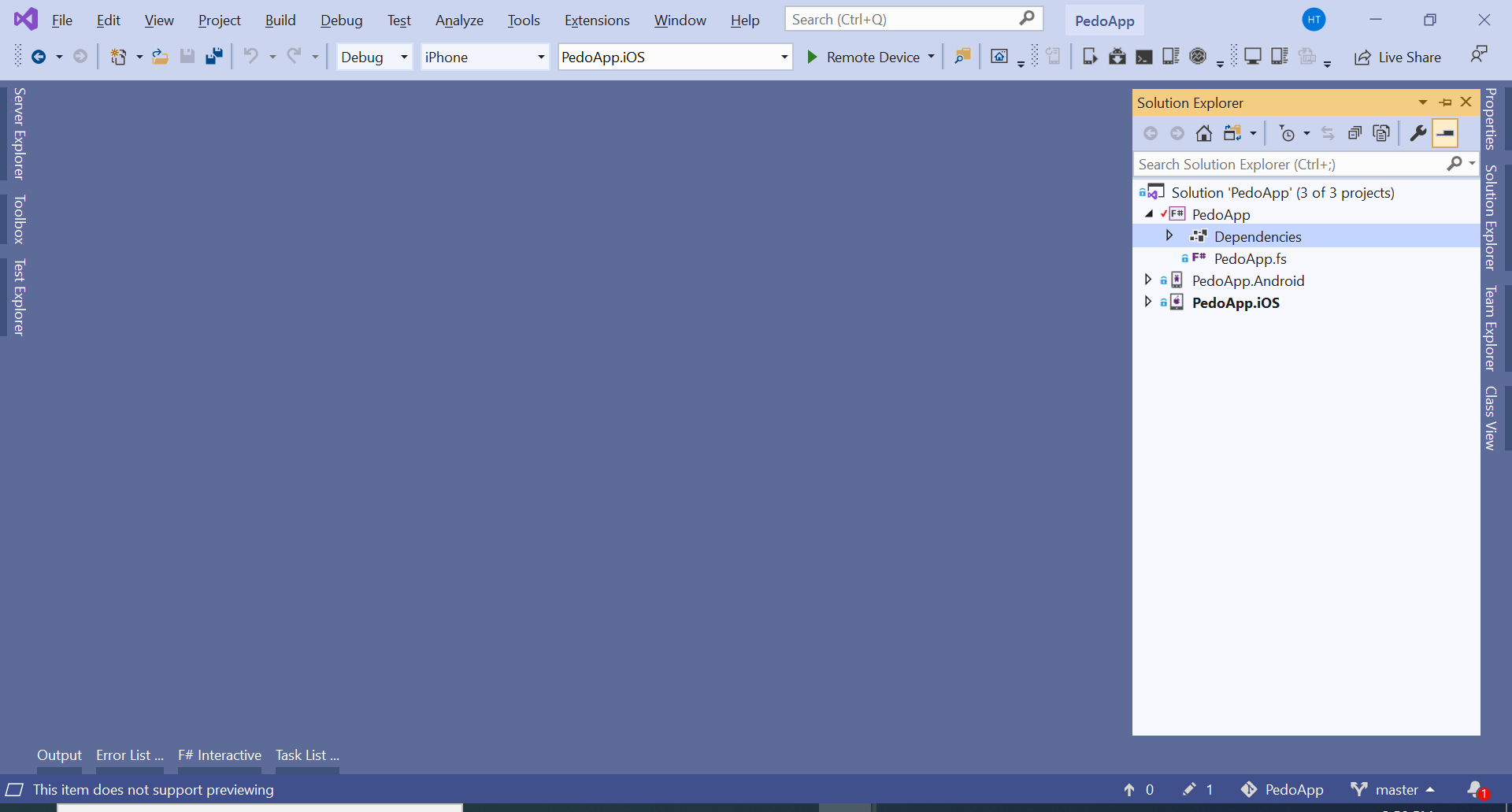
* Press Windows+R
* Type “cmd”



* Press Enter
* Input “dotnet new -i Fabulous.XamarinForms.Templates” and press Enter 

Starting a new project:

* Input “cd %HOMEPATH%” and press Enter
* Input “dotnet new fabulous-xf-app -n PedoApp” and press Enter
* Input “start PedoApp/PedoApp.sln” and press Enter
* Wait for Visual Studio to load
* Press the triangle to the left of “PedoApp”
* Select “PedoApp.fs” (remember this as “PedoApp.fs”)



Structure of a program:

// Copyright 2018-2019 Fabulous contributors. See LICENSE.md for license.

namespace PedoApp

We “open” to import code from other people

open System.Diagnostics

open Fabulous

open Fabulous.XamarinForms

open Fabulous.XamarinForms.LiveUpdate

open Xamarin.Forms

int: Integer (Whole number), e.g. -3, -2, -1, 0, 1, 2, 3…  
bool: Boolean (True/False) “3 = 3” “2.3 = 2.3” are True “3 = 4” “3 ≠ 3” are False

module App =

type Model =

{ Count : int

Step : int

“Model” is the status of the app (nouns)

Count: Current number in counter

Step: Change in number every increment   
TimerOn: Whether number increments automatically

TimerOn: bool }

type Msg =

| Increment

| Decrement

| Reset

“Msg”, aka “Message”, is an event that can cause the “Model” to change (verbs)

| SetStep of int

| TimerToggled of bool

| TimedTick

“initModel”, aka Initial Model, is the status of the app after launch

let initModel = { Count = 0; Step = 1; TimerOn=false }

let init () = initModel, Cmd.none

“init”: Combines “initModel” and a “Cmd” that represents additional “Msg”s after app launch

“Cmd”, aka “Command”, decide whether additional “Msg”s are sent after a “Msg” is handled

Cmd.none: We don’t cause additional “Msg”s after app launch

let timerCmd =

async { do! Async.Sleep 200

return TimedTick }

|> Cmd.ofAsyncMsg

timerCmd: A “Cmd” that cause TimedTick (a “Msg”) after 200 milliseconds

“update”: How each “Msg” changes “Model” and optionally causes more “Msg”s through “Cmd”s

let update msg model =

match msg with

| Increment -> { model with Count = model.Count + model.Step }, Cmd.none

| Decrement -> { model with Count = model.Count - model.Step }, Cmd.none

| Reset -> init ()

| SetStep n -> { model with Step = n }, Cmd.none

| TimerToggled on -> { model with TimerOn = on }, (if on then timerCmd else Cmd.none)

| TimedTick ->

if model.TimerOn then

{ model with Count = model.Count + model.Step }, timerCmd

else

model, Cmd.none

“view”: Convert “Model” to an actual User Interface

let view (model: Model) dispatch =

View.ContentPage(

content = View.StackLayout(padding = Thickness 20.0, verticalOptions = LayoutOptions.Center,

children = [

View.Label(text = sprintf "%d" model.Count, horizontalOptions = LayoutOptions.Center, width=200.0, horizontalTextAlignment=TextAlignment.Center)

View.Button(text = "Increment", command = (fun () -> dispatch Increment), horizontalOptions = LayoutOptions.Center)

View.Button(text = "Decrement", command = (fun () -> dispatch Decrement), horizontalOptions = LayoutOptions.Center)

View.Label(text = "Timer", horizontalOptions = LayoutOptions.Center)

View.Switch(isToggled = model.TimerOn, toggled = (fun on -> dispatch (TimerToggled on.Value)), horizontalOptions = LayoutOptions.Center)

View.Slider(minimumMaximum = (0.0, 10.0), value = double model.Step, valueChanged = (fun args -> dispatch (SetStep (int (args.NewValue + 0.5)))), horizontalOptions = LayoutOptions.FillAndExpand)

View.Label(text = sprintf "Step size: %d" model.Step, horizontalOptions = LayoutOptions.Center)

View.Button(text = "Reset", horizontalOptions = LayoutOptions.Center, command = (fun () -> dispatch Reset), commandCanExecute = (model <> initModel))

]))

// Note, this declaration is needed if you enable LiveUpdate

let program = XamarinFormsProgram.mkProgram init update view

“program”: Combine “init”, “update” and “view” into an app

type App () as app =

inherit Application ()

let runner =

App.program

#if DEBUG

Run the “program” on app launch

|> Program.withConsoleTrace

#endif

|> XamarinFormsProgram.run app

#if DEBUG

// Uncomment this line to enable live update in debug mode.

// See https://fsprojects.github.io/Fabulous/Fabulous.XamarinForms/tools.html#live-update for further instructions.

//

Live Update: We don’t need it right now

//do runner.EnableLiveUpdate()

#endif

// Uncomment this code to save the application state to app.Properties using Newtonsoft.Json

// See https://fsprojects.github.io/Fabulous/Fabulous.XamarinForms/models.html#saving-application-state for further instructions.

#if APPSAVE

Reusing “Model” from previous launches: We don’t need it right now

let modelId = "model"

override \_\_.OnSleep() =

let json = Newtonsoft.Json.JsonConvert.SerializeObject(runner.CurrentModel)

Console.WriteLine("OnSleep: saving model into app.Properties, json = {0}", json)

app.Properties.[modelId] <- json

override \_\_.OnResume() =

Console.WriteLine "OnResume: checking for model in app.Properties"

try

match app.Properties.TryGetValue modelId with

| true, (:? string as json) ->

Console.WriteLine("OnResume: restoring model from app.Properties, json = {0}", json)

let model = Newtonsoft.Json.JsonConvert.DeserializeObject<App.Model>(json)

Console.WriteLine("OnResume: restoring model from app.Properties, model = {0}", (sprintf "%0A" model))

runner.SetCurrentModel (model, Cmd.none)

| \_ -> ()

with ex ->

App.program.onError("Error while restoring model found in app.Properties", ex)

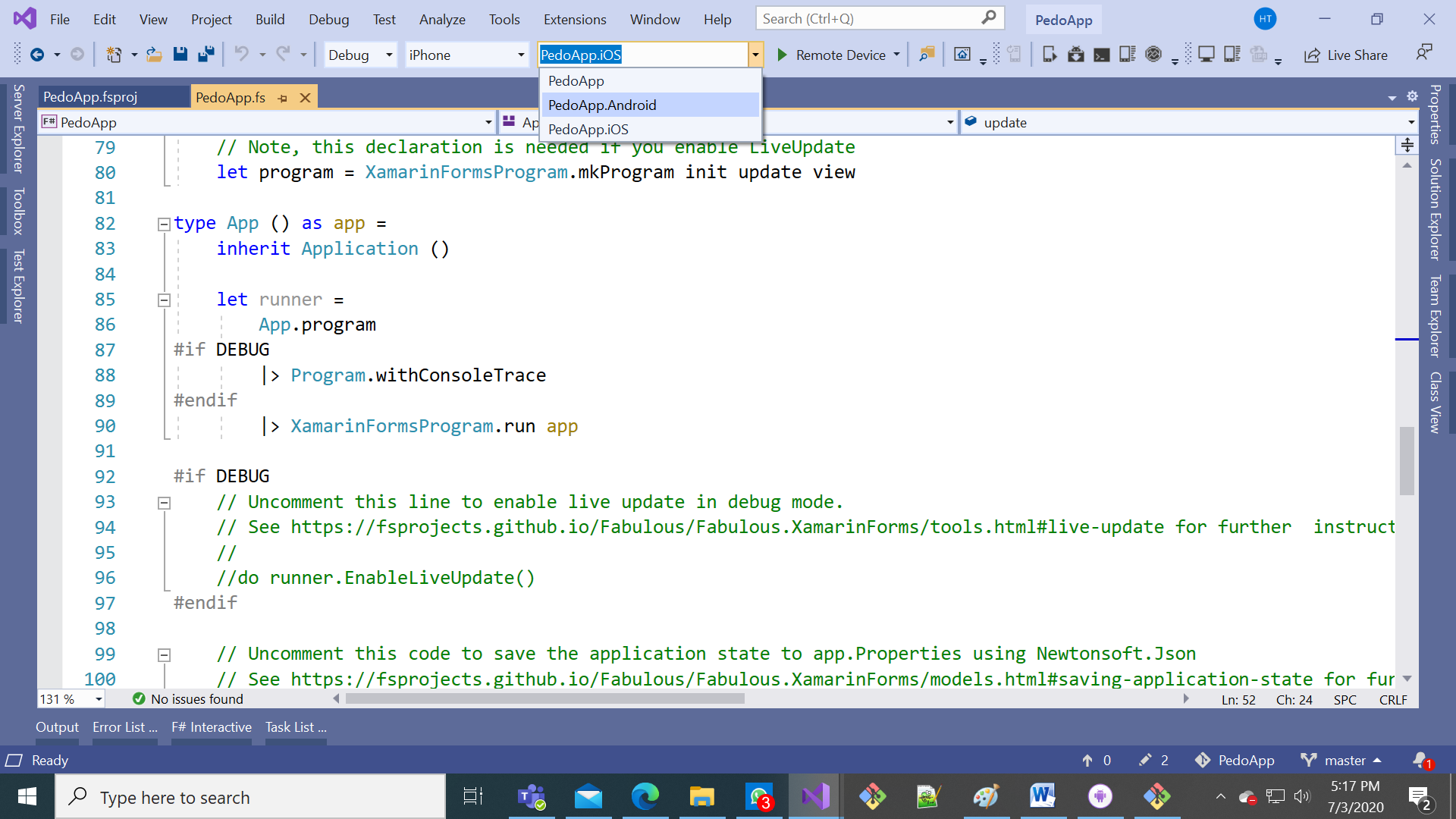
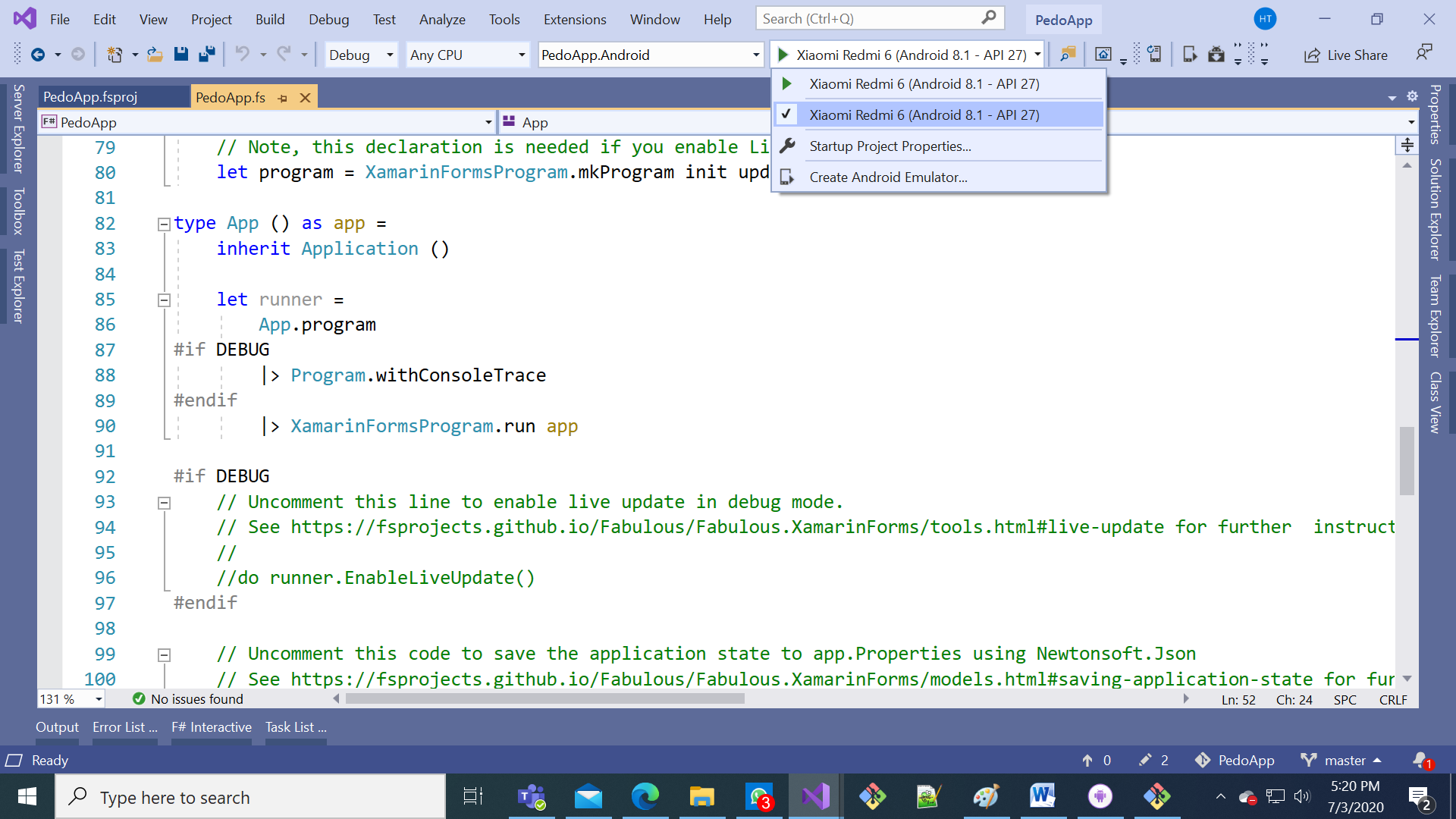
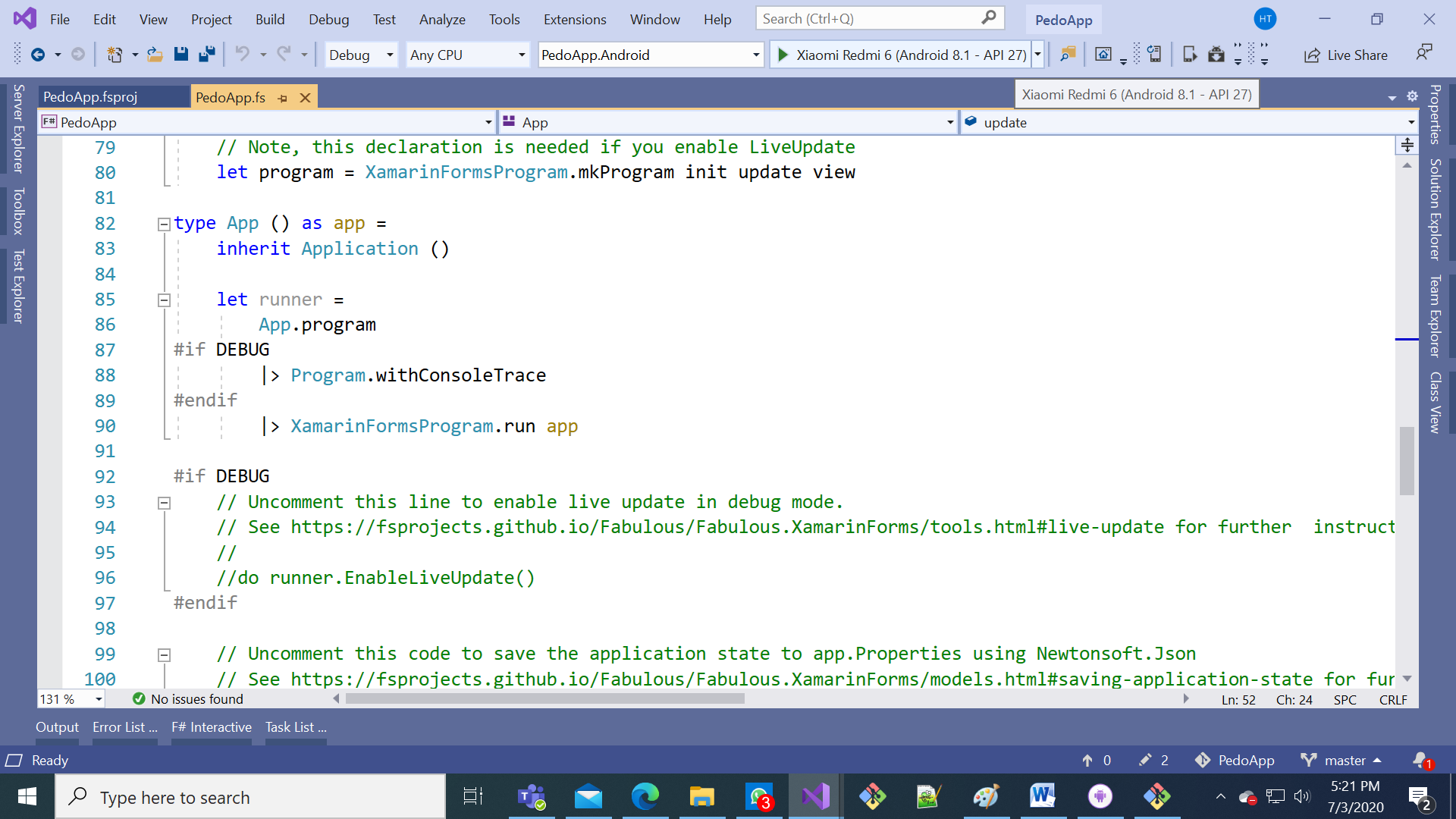
override this.OnStart() =

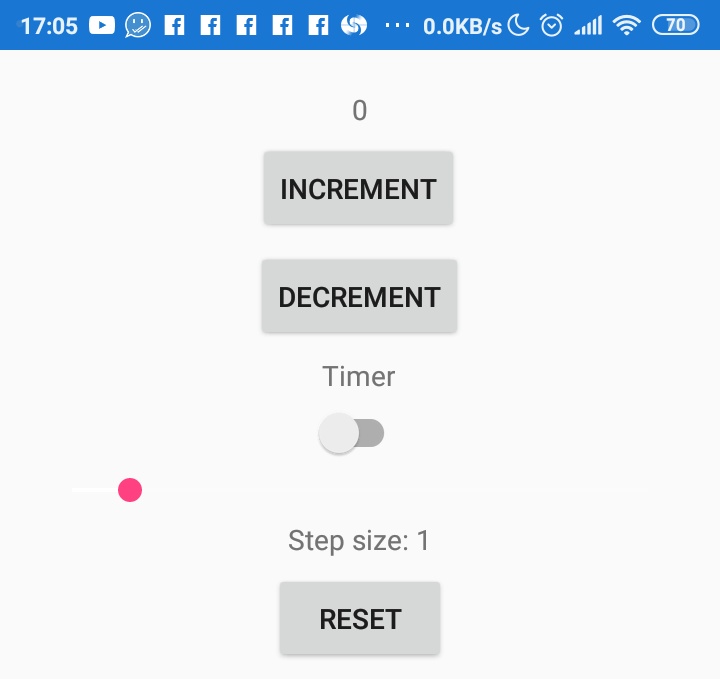
Console.WriteLine "OnStart: using same logic as OnResume()"

this.OnResume()

#endif

Run the app (Android):

* Enable USB Debugging in your phone as in <https://www.kingoapp.com/root-tutorials/how-to-enable-usb-debugging-mode-on-android-5-lollipop.htm>
* Plug your phone into the computer via a USB cable
* Select “PedoApp.Android” as the startup project  
  
* Select your device from the dropdown menu
* Press the green arrow to start debugging 
* Watch out for installation dialogs on your phone and accept all the dialogs
* The app should launch by itself on your phone



Run the app (iOS): [To Be Added]

Adding a pedometer to the app:

* Remove red lines, add green lines in “PedoApp.fs”

...

open Xamarin.Forms

module App =

type Pedometer =

abstract Step : IEvent<int>

type Model =

{ Count : int

Step : int

TimerOn: bool }

TimerOn: bool

Pedometer : int }

type Msg =

| Increment

...

| SetStep of int

| TimerToggled of bool

| TimedTick

| PedometerUpdated of int

let initModel = { Count = 0; Step = 1; TimerOn=false }

let initModel = { Count = 0; Step = 1; TimerOn=false; Pedometer = 0 }

let init () = initModel, Cmd.none

let init () =

initModel, Cmd.ofSub (fun dispatch -> DependencyService.Get<Pedometer>().Step.Add(PedometerUpdated >> dispatch))

let timerCmd =

async { do! Async.Sleep 200

...

{ model with Count = model.Count + model.Step }, timerCmd

else

model, Cmd.none

| PedometerUpdated p -> { model with Pedometer = p }, Cmd.none

let view (model: Model) dispatch =

View.ContentPage(

...

View.Slider(minimumMaximum = (0.0, 10.0), value = double model.Step, valueChanged = (fun args -> dispatch (SetStep (int (args.NewValue + 0.5)))), horizontalOptions = LayoutOptions.FillAndExpand)

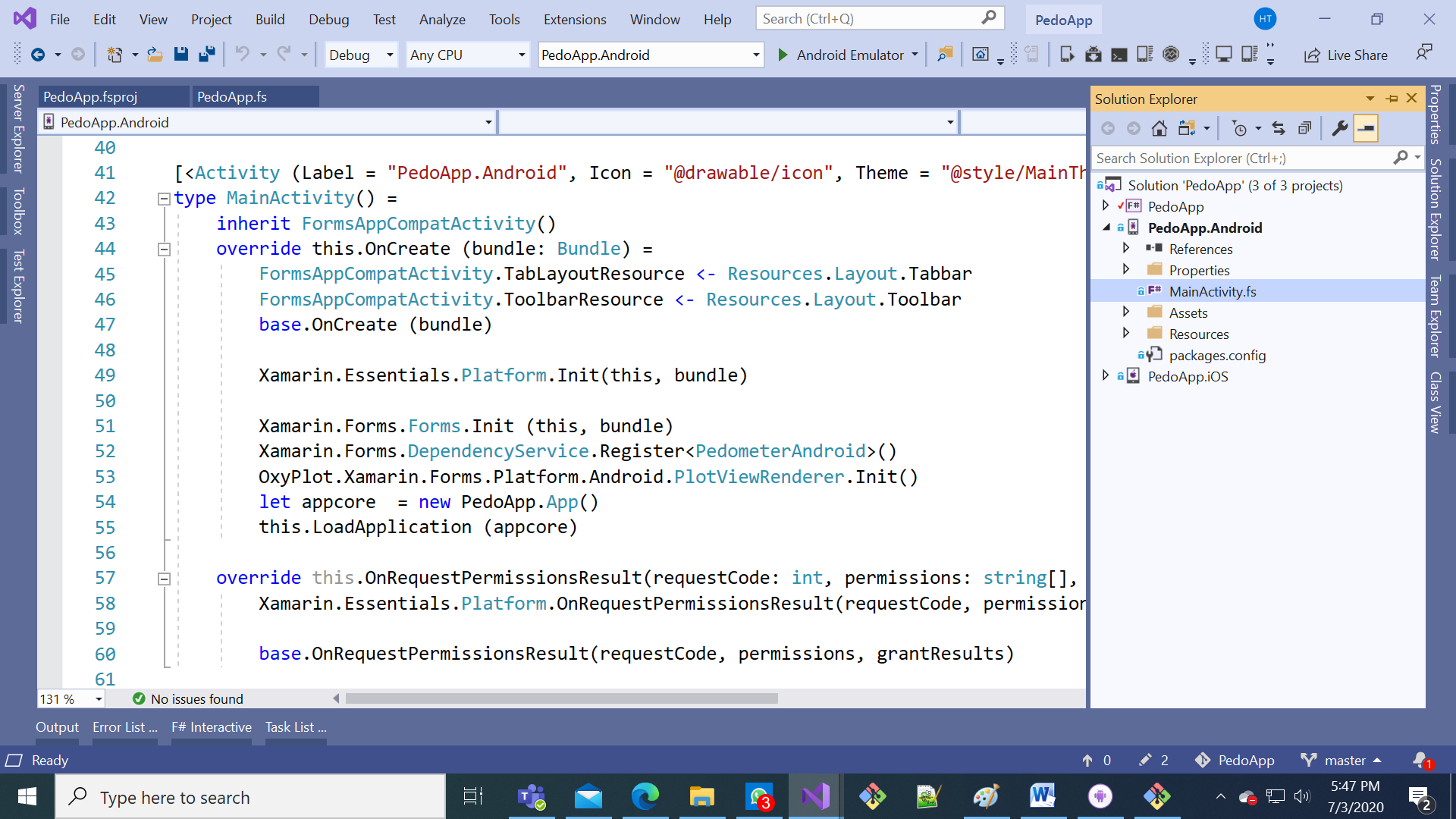
View.Label(text = sprintf "Step size: %d" model.Step, horizontalOptions = LayoutOptions.Center)

View.Button(text = "Reset", horizontalOptions = LayoutOptions.Center, command = (fun () -> dispatch Reset), commandCanExecute = (model <> initModel))

View.Label(text = sprintf "Pedometer: %d" model.Pedometer)

]))

// Note, this declaration is needed if you enable LiveUpdate

Open “MainActivity.fs” under “PedoApp.Android” (remember this as “MainActivity.fs”): 

* Remove red lines, add green lines in “MainActivity.fs”

...

open Android.OS

open Xamarin.Forms.Platform.Android

type PedometerListener(onChanged) =

inherit Java.Lang.Object()

interface Android.Hardware.ISensorEventListener with

member \_.OnAccuracyChanged(\_, \_) = ()

member \_.OnSensorChanged e =

// steps since device boot

onChanged <| int e.Values.[0]

type PedometerAndroid() =

let manager =

Context.SensorService

|> Xamarin.Essentials.Platform.CurrentActivity.GetSystemService

:?> Android.Hardware.SensorManager

let sensor = manager.GetDefaultSensor Android.Hardware.SensorType.StepCounter

let event = Event<int>()

do manager.RegisterListener(new PedometerListener(event.Trigger), sensor, Android.Hardware.SensorDelay.Fastest)

|> ignore

interface PedoApp.App.Pedometer with member \_.Step = event.Publish

[<UsesPermission("android.permission.ACTIVITY\_RECOGNITION")>]

[<UsesFeature(Name=Android.Hardware.Sensor.StringTypeStepCounter, Required=true)>]

do ()

[<Activity (Label = "PedoApp.Android", Icon = "@drawable/icon", Theme = "@style/MainTheme", MainLauncher = true, ConfigurationChanges = (ConfigChanges.ScreenSize ||| ConfigChanges.Orientation))>]

type MainActivity() =

inherit FormsAppCompatActivity()

...

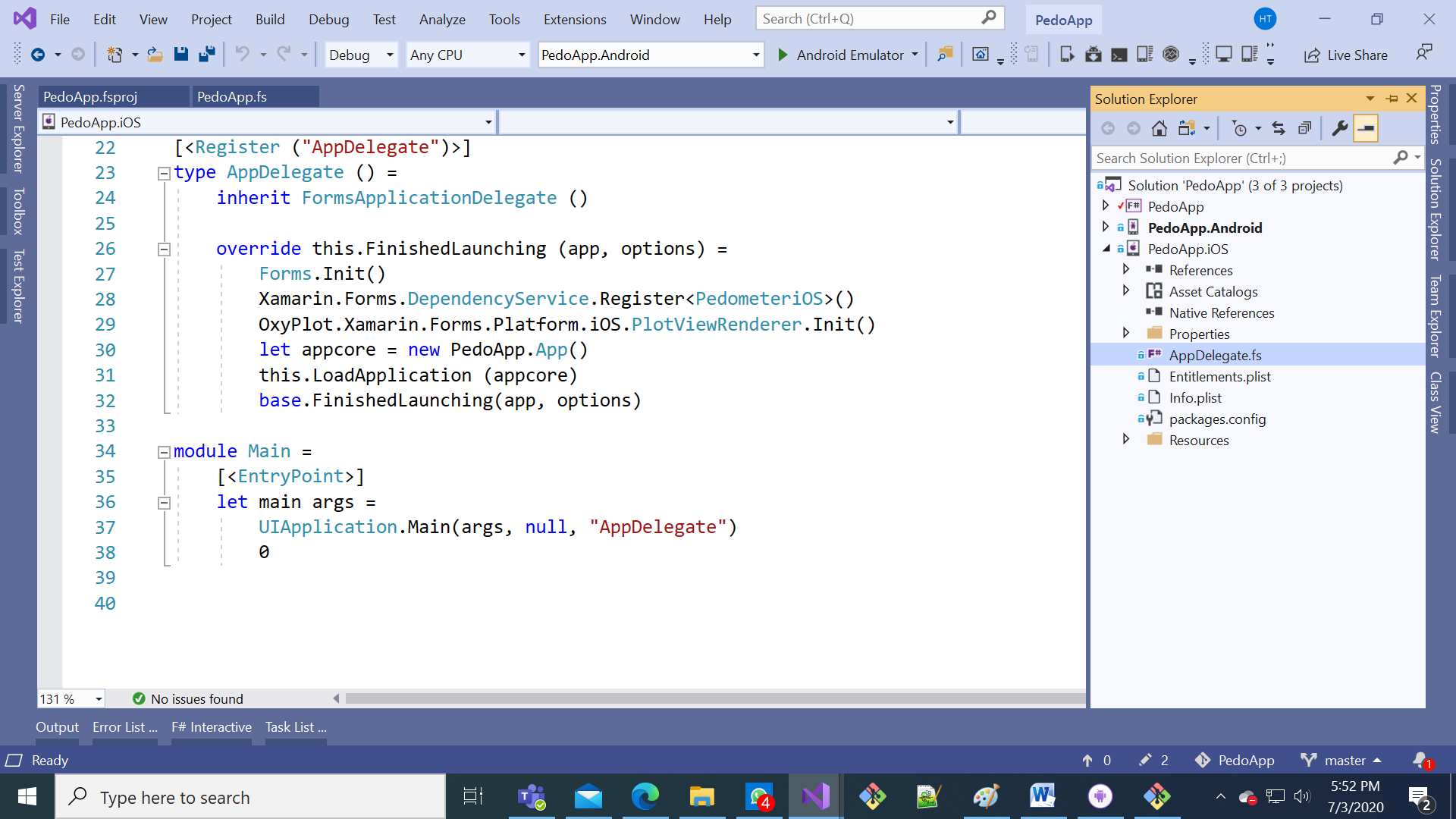
Xamarin.Essentials.Platform.Init(this, bundle)

Xamarin.Forms.Forms.Init (this, bundle)

Xamarin.Forms.DependencyService.Register<PedometerAndroid>()

let appcore = new PedoApp.App()

this.LoadApplication (appcore)

Open “AppDelegate.fs” under “PedoApp.iOS” (remember this as “AppDelegate.fs”): 

* Remove red lines, add green lines in “MainActivity.fs”

...

open Xamarin.Forms

open Xamarin.Forms.Platform.iOS

type PedometeriOS() =

let pedometer = new CoreMotion.CMPedometer()

let event = Event<int>()

// steps from midnight

do pedometer.StartPedometerUpdates(DateTime.Today.ToNSDate(),

Action<\_, \_>(fun data \_ -> event.Trigger data.NumberOfSteps.Int32Value))

interface PedoApp.App.Pedometer with member \_.Step = event.Publish

[<Register ("AppDelegate")>]

type AppDelegate () =

inherit FormsApplicationDelegate ()

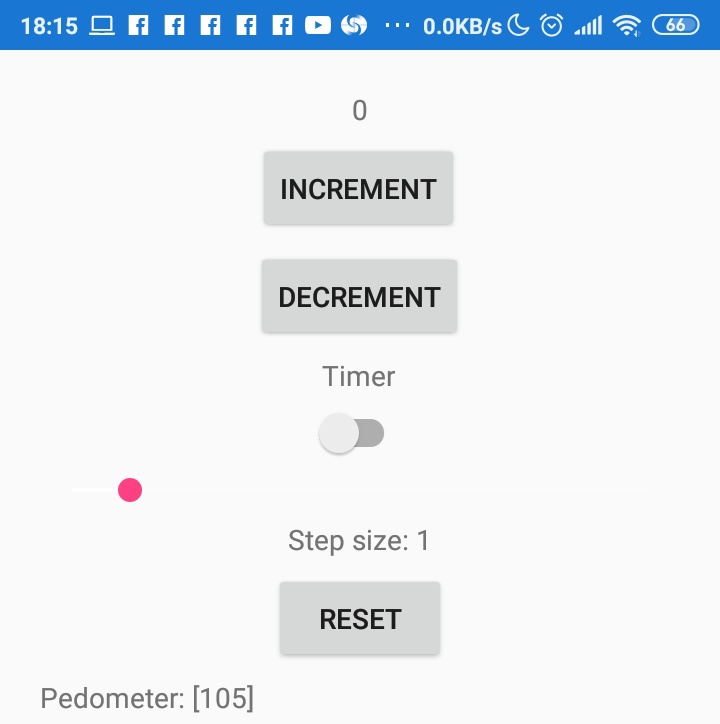
...

override this.FinishedLaunching (app, options) =

Forms.Init()

Xamarin.Forms.DependencyService.Register<PedometeriOS>()

Run the app:



The data updates every minute because the sensor stabilizes the data to improve accuracy

Solving the iOS crash & Checking pedometer availability: [To be added]

Graphing the data: [To be added]