# UI设计：2种方法

MainPage.xaml里面

<Label x:Name="Label1" // Label相当于一个TextView

Text="Hello World!" // 内容

HorizontalOptions="Center" // 位置

VerticalOptions="Center">

</Label>

<Slider x:Name="slider1" // 滑块，名字叫slider

Maximum="20"> // 滑到头是20

</Slider>

<BoxView // 分割线

HeightRequest="1" BackgroundColor="Gray">

</BoxView>

Text各种属性

Text="Welcome to Xamarin!" // 文本内容

HorizontalOptions="Center" // 水平居中

TextColor="White" // 字的颜色

FontSize="20" // 字号

BackgroundColor="Black" // 背景颜色

BackgroundColor="#009682" // 颜色可以用16进制数表示

Opacity="0.5" // 透明度

FontAttributes="Bold" // 加粗

HorizontalTextAlignment="Center" // 在背景框里居中

VerticalTextAlignment="Center"

居中和背景框里居中的区别：

如果定义了背景框，再使用HorizontalOptions="Center"的话，背景框就没了。

如果使用HorizontalTextAlignment="Center"，则背景框还在，文字在背景框中间

Padding 和背景颜色在xaml的写法

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

BackgroundColor="#4664AA"

Padding="20"

x:Class="Layout.MainPage">

不同平台下的Padding写法

<ContentPage.Padding>

<OnPlatform x:TypeArguments="Thickness"

iOS = "0, 20, 0, 0"

Android = "0">

</OnPlatform>

</ContentPage.Padding>

在MainPage.xaml.cs里面

public MainPage()

{

InitializeComponent();

Content = new Label

{

Text = "Hello World!"

};

}

在.xaml中定义了名字之后也可以在.cs里面利用Name属性直接修改

public MainPage()

{

InitializeComponent();

Label1.Text = "Hello";

}

还可以加上Padding（类似于Margin）

public MainPage()

{

InitializeComponent();

Padding = new Thickness

{

Top = 20, Bottom = 20, Left = 20, Right = 20

};

}

一般写为

Padding = new OnPlatform<Thickness>

{

Android = new Thickness(0),

iOS = new Thickness(0, 20, 0, 0) // left, top, right, bottom

};

# Data Binding

在.xaml里面

<Label x:Name="Label1"

Text="{Binding Source={x:Reference slider1}, Path=Value, StringFormat= 'Value is {0:F2}'}"

HorizontalOptions="Center"

VerticalOptions="Center">

</Label>

Binding 表示DataBinding

Source={x:Reference slider1} 表示连接的是slider1

Path=Value 表示获取Slider里面的Value属性

StringFormat= 'Value is {0:F2}' 表示输出的字符串

例如FontSize="{Binding Source={x:Reference slider1}, Path=Value}" 中，字体大小本身就是数值就可以直接用来改变字体大小。

同一个View的多个属性都与同一个View连接时，可以直接写BindingContext="{x:Reference slider1}"，不必重复写：

<Label x:Name="Label1"

BindingContext="{x:Reference slider1}"

Text="{Binding Path=Value, StringFormat= 'Value is {0:F2}'}"

FontSize="{Binding Path=Value}"

HorizontalOptions="Center"

VerticalOptions="Center">

</Label>

如果在一个页面里都用到同一个View的，可以直接写在Layout里面：

<StackLayout BindingContext="{x:Reference slider1}" VerticalOptions="Center"> // BindingContext写在这

<Label x:Name="Label1"

Text="{Binding Path=Value, StringFormat= 'Value is {0:F2}'}"

HorizontalOptions="Center"

VerticalOptions="Center">

</Label>

<Label x:Name="Label2"

Text="This is Text 2."

Opacity="{Binding Path=Value}" // 透明度 0 - 1

HorizontalOptions="Center"

VerticalOptions="Center">

</Label>

<Slider x:Name="slider1"

Maximum="1">

</Slider>

</StackLayout>

绑定cs和xaml数据

在ContentPage里面加上

xmlns:vm="clr-namespace:项目名.文件夹名"

在ContentPage和ContentPage.Content之间写上

<ContentPage.BindingContext>

<vm:类的名字></vm:类的名字>

</ContentPage.BindingContext>

在cs文件里InitializeComponent();后面加上

BindingContext = new DataViewModel; // 或者BindingContext = m; 绑定传入的数据

在xaml文件里的Text部分绑定上内容

Text="{Binding TEMPERATURE}"

其中TEMPERATURE是DataViewModel里的一个函数。下面详细讲：

在DataViewModel里先建立必要的内容，比如

public string temperature = "0"；

然后写上对应的get和set

public string TEMPERATURE

{

get { return temperature; } // View从这里读取

set { temperature = value; } // 改变变量从这里改变

}

变量的改变方式就是直接赋值，例如

dataviewmodel.TEMPERATURE = "20";

在xaml文件里必须写一个callback函数，每次值改变时调用，用来更新get到的文字：

先改DataViewModel的继承关系

public class DataViewModel : INotifyPropertyChanged

会提示创建一个Handler

public event PropertyChangedEventHandler PropertyChanged;

利用Handler我们可以写一个变化时候的CallBack函数

public void OnPropertyChanged([CallerMemberName] string name = "")

{

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(name));

}

// 这里[CallerMemberName]可以自动获取它所在的函数，比如TEMPERATURE。不加这个，就要在string name里加上函数的名字。比如

public string TEMPERATURE

{

get { return temperature; }

set

{

temperature = value;

OnPropertyChanged(); // 这里就更新了TEMPERATURE自己

OnPropertyChanged(PRESSURE); // 这里就时在temperature赋值时调用了其他的函数

}

}

最终DataViewModel里面的大部分函数都是这个形式的。

# Layout

在.xaml里面

StackLayout的属性

<StackLayout

Orientation = "Vertical" // 排布方式：竖直

VerticalOptions = "Center" // 竖直方向：居中

HorizontalOptions = "Center" // 水平方向：居中

Spacing = "10" // View和View之间的空间

Padding = "10"> // Padding

往页面里加东西

<StackLayout

Orientation="Horizontal"

VerticalOptions="Center">

<Label Text="Label 1"></Label>

<Label Text="Label 2"></Label>

<Label Text="Label 3"></Label>

<Label Text="Label 4"></Label>

</StackLayout>

在.cs里面

var stakelayout = new StackLayout

{

VerticalOptions = LayoutOptions.Center,

Orientation = StackOrientation.Horizontal

};

stakelayout.Children.Add(new Label { Text = "Label1" });

stakelayout.Children.Add(new Label { Text = "Label2" });

stakelayout.Children.Add(new Label { Text = "Label3" });

stakelayout.Children.Add(new Label { Text = "Label4" });

Content = stakelayout;

这里stakelayout是一个StakeLayout的对象，用.Children.Add来加东西，最后把Content设置成stakelayout。

# GridLayout

定义Grid

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="\*"></RowDefinition>

<RowDefinition Height="\*"></RowDefinition>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"></ColumnDefinition>

<ColumnDefinition Width="\*"></ColumnDefinition>

</Grid.ColumnDefinitions>

// 内容

</Grid>

其中\*表示把屏幕分开

<Grid.RowDefinitions>

<RowDefinition Height="2\*"></RowDefinition> // 上面的占2/3

<RowDefinition Height="\*"></RowDefinition> // 下面的占1/3

</Grid.RowDefinitions>

还可以写auto

<RowDefinition Height="Auto"></RowDefinition> // 意味着这个grid的高度由内容决定

内容

<Grid>

// 定义Grid

<Label Text="Top Left" BackgroundColor="#009682" FontSize="30"

HorizontalOptions="Center" VerticalOptions="Center"

TextColor="White"

Grid.Row="0" Grid.Column="0"> // 指定位置

</Label>

// 其他内容

</Grid>

# Button点击

以一个拨号页面为例

<Label x:Name="phonenumber"

Text="1234" FontSize="20"

FontAttributes="Bold" HorizontalOptions="Center"

VerticalOptions="Center"

Grid.Row="0" Grid.ColumnSpan="3"></Label>

<Button x:Name="button1" Text="1" FontAttributes="Bold" HorizontalOptions="Center"

VerticalOptions="Center"

Clicked="Handle\_Clicked" // 此处设置了在点击时触发Handle\_Clicked函数

Grid.Row="1" Grid.Column="0"></Button>

函数在xaml.cs里面

namespace Griddemo

{

public partial class MainPage : ContentPage // 在这里面加

{

void Handle\_Clicked(System.Object sender, System.EventArgs e) // 这个函数

{

var numberButton = (sender as Button); // numberButton就有了button的信息

var number = numberButton.Text; // 取出button的Text内容

phonenumber.Text += number; // 拼接在电话号码后面

}

public MainPage()

{

InitializeComponent();

phonenumber.Text = ""; // 初始化为空字符串

}

}

}

当然可以增加一个删除按钮

<Button x:Name="buttonl" Text="L" FontAttributes="Bold" HorizontalOptions="Center"

VerticalOptions="Center" Clicked="Handle\_Clicked\_Delete"

Grid.Row="4" Grid.Column="2"></Button>

触发Handle\_Clicked\_Delete函数

void Handle\_Clicked\_Delete(System.Object sender, System.EventArgs e)

{

if (phonenumber.Text.Length > 0)

{

phonenumber.Text = phonenumber.Text.Substring(0, phonenumber.Text.Length - 1);

}

}

# AbsolutLayout

<AbsoluteLayout>

<BoxView BackgroundColor="Black" Opacity="0.6"

AbsoluteLayout.LayoutBounds="0,0,1,1"

AbsoluteLayout.LayoutFlags="All"></BoxView>

<Label Text="Hello World" TextColor="Black" BackgroundColor="White" Opacity="0.6"

HorizontalOptions="Center" VerticalOptions="Center"

AbsoluteLayout.LayoutBounds="0.5,0.3,100,100"

AbsoluteLayout.LayoutFlags="PositionProportional"></Label>

</AbsoluteLayout>

定义位置有2种方法：比例和像素。

位置百分比，长度是像素

AbsoluteLayout.LayoutBounds="0.5,0.3,100,100" // x位置，y位置，x长度，y长度

AbsoluteLayout.LayoutFlags="PositionProportional" // 位置是百分比，长度是像素值

都是百分比

AbsoluteLayout.LayoutBounds="0,0,1,1" // x位置，y位置，x长度，y长度

AbsoluteLayout.LayoutFlags="All" // 都是百分比

位置+宽度是百分比

AbsoluteLayout.LayoutBounds="0.5,0.2,0.5,100" // x位置，y位置，x长度，y长度

AbsoluteLayout.LayoutFlags="PositionProportional, WidthProportional" // 位置+宽度是百分比

# Image图片

导入图片

Android放在Resource/Drawable里面

iOS放在Resource里面

加载图片

<Image x:Name="image1" Source="Photo.jpg"></Image>

除此之外还有Aspect

<Image Aspect="AspectFit" Source="Photo.jpg"></Image>

AspectFill 不改变比例，填满空间，图像有部分不被展示

AspectFit 不改变比例，在空间中放到最大，会有部分空白

Fill 改变比例以填满空间

# Entry输入框

<Entry Placeholder="Chat" // 输入框背景内容 hint

Keyboard="Chat" // 键盘类型

IsPassword="True" // 是否显示输入内容

TextChanged="Entry\_TextChanged" // 内容发生改变时的事件

Completed="Entry\_Completed"> // 内容完成时的事件

</Entry>

键盘类型一般有：plain，numeric

TextChanged用法举例：

void Entry\_TextChanged(System.Object sender, Xamarin.Forms.TextChangedEventArgs e)

{

Text1.Text = e.NewTextValue;

}

Completed

void Entry\_Completed(System.Object sender, System.EventArgs e)

{

var entry = sender as Entry;

Text1.Text = entry.Text;

}

# Tabbed Page (多栏页面)

1. 右键单击项目（.Forms），添加，新建文件，Forms，窗体ContentPage XAML，在下面起名字，新建。

2. 在MainPage.xaml里面，把最外层的ContentPage改成TabbedPage

3. 加上namespace：xmlns:local="clr-namespace:项目名字"

<?xml version="1.0" encoding="utf-8" ?>

<TabbedPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:local="clr-namespace:TabbedPageDemo" // 这里

x:Class="TabbedPageDemo.MainPage">

</TabbedPage>

4. 在.xaml.cs里面修改继承

namespace TabbedPageDemo

{

public partial class MainPage : TabbedPage // 这里

{

public MainPage()

{

InitializeComponent();

}

}

}

5. 添加页面

<?xml version="1.0" encoding="utf-8" ?>

<TabbedPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:local="clr-namespace:TabbedPageDemo"

x:Class="TabbedPageDemo.MainPage">

<local:Page1 Title="page1" ></local:Page1>

<local:Page2 Title="page2" ></local:Page2>

<local:Page3 Title="page3" ></local:Page3>

</TabbedPage>

这里Page1, Page2, Page3是我创建的Page的名字，Title是在APP上显示的页面名字。

在子页面中，放置内容方式与以前相同，例如

<?xml version="1.0" encoding="UTF-8"?>

<ContentPage

xmlns=http://xamarin.com/schemas/2014/forms

xmlns:x=http://schemas.microsoft.com/winfx/2009/xaml

x:Class="TabbedPageDemo.Page1">

<ContentPage.Content>

<Image Aspect="AspectFill" Source="Photo.jpg"></Image> // 放一张照片

</ContentPage.Content>

</ContentPage>

# StackLayout.GestureRecognizers

以切换页面为例

1. 先建立页面DemoPage1

2. 在MainPage里建立一个放内容的东西 ContentView

<ContentView x:Name="Inhalt"></ContentView>

3. .xaml里面中：在需要点击的StackLayout里面加入点击触发的函数

<StackLayout.GestureRecognizers>

<TapGestureRecognizier Tapped= "Demo\_Tapped"> // Demo\_Tapped是个函数

</StackLayout.GestureRecognizers>

4. 在.xaml.cs里面

Void Demo\_Tapped(object sender, System.EventArgs e)

{

var page = new DemoPage1(); // 获取DemoPage1

Inhalt.Content = page.Content; // 将page的内容赋给Inhalt的内容

}

# Navigation

在button上设置一个Clicked

在cs文件里面加入导航

void Button\_Clicked(System.Object sender, System.EventArgs e)

{

Navigation.PushModalAsync(new MyPage());

}

在另一个页面上加上返回

void Button\_Clicked(System.Object sender, System.EventArgs e)

{

Navigation.PopModalAsync();

}

# ListView

在xaml里面建立一个ListView

<ListView x:Name="listView" // 名字

ItemSelected="listView\_ItemSelected" // 点击后的效果

ItemsSource="{Binding .}" /> // 先后面指定

还可以指定ListView的具体形式，比如：

<ListView x:Name="listView"> // 上文的ListView属性都写在这行里

<ListView.ItemTemplate>

<DataTemplete>

<TextCell Text="{Binding Name}" Detail="Binding Weight"/>

</DataTemplete>

</ListView.ItemTemplate>

</ListView>

TextCell有4个属性，Text， Detail和他们的颜色，分别是列表里面的内容和每一行下面的小注释。

在cs文件里

public MainPage()

{

InitializeComponent();

listView.ItemsSource = new [] { "a", "b", "c" }; // 定义一个ListView内容

}

void listView\_ItemSelected(System.Object sender, Xamarin.Forms.SelectedItemChangedEventArgs e)

{

if (e.SelectedItem == null) return; // 没选中时没事

DisplayAlert("Tapped", e.SelectedItem + " was tapped", "OK"); // 显示一个框

((ListView)sender).SelectedItem = null; // 重置

}

e.SelectedItem就是内容

sender就是ListView

# 蓝牙BLE

在项目/依赖项/NuGet上点右键，点击管理NuGet包，右上角搜索Plugin.BLE添加作者为Adrian的一项。

在cs页面中添加IBluetoothLE

namespace AirCase

{

public partial class BluetoothConnection : ContentPage

{

IBluetoothLE

public BluetoothConnection()

{

InitializeComponent();

}

}

}

然后点击IBluetoothLE报错的前面，自动添加所需的包

using Plugin.BLE.Abstractions.Contracts;

之后再写一些东西

namespace AirCase

{

public partial class BluetoothConnection : ContentPage

{

IBluetoothLE ble;  // 这里

IAdapter adapter; // 这里

ObservableCollection<IDevice> deviceList; // 这里

public BluetoothConnection()

{

InitializeComponent();

ble = CrossBluetoothLE.Current; // 这里

adapter = CrossBluetoothLE.Current.Adapter; // 这里

deviceList = new ObservableCollection<IDevice>(); // 这里

}

}

}

在.Android/Properities/AndroidManifest.xml里面选上

AccessCoarseLocation

AccessFineLocation

Bluetooth

BluetoothAdmin

但是即便如此app也没有权限，要导入

Xamarin.Essentials

Plugin.Permissions

包，并且在代码里写上

public BluetoothConnection()

{

InitializeComponent();

ble = CrossBluetoothLE.Current;

adapter = CrossBluetoothLE.Current.Adapter;

deviceList = new ObservableCollection<IDevice>();

btlist.ItemsSource = deviceList;

RequestPermissions(); // 非常关键！ 给app本地权限

}

private async void RequestPermissions()

{

var status = await CrossPermissions.Current.RequestPermissionAsync<LocationAlwaysPermission>();

Console.WriteLine("requested location permission: " + status.ToString());

}

获取列表：

检查蓝牙状态：

var state = ble.State;

if (state.ToString() == "Off")

{

await DisplayAlert("Bluetooth State", "Bluetooth is off, please turn on the Bluetooth.", "OK");

}

else

{

Console.WriteLine("BLE STATUS IS: " + state.ToString());

}

获取列表并刷新到ListView里面

btlist.ItemsSource = deviceList; // 指定ListView的资源，Device列表，每一个就是一个Device

deviceList.Clear();

adapter.DeviceDiscovered += (s, a) =>

{

if (a.Device.Name == null | a.Device.Id == null) { } // 检测发现的目标名字和ID

else

{

deviceList.Add(a.Device);

Console.WriteLine(deviceList.Count.ToString() + " devices was found.");

}

};

if (!ble.Adapter.IsScanning)

{

await adapter.StartScanningForDevicesAsync();

}

（ListView）列表：

<ListView x:Name="btlist"

ItemSelected="btlist\_ItemSelected">

<ListView.ItemTemplate>

<DataTemplate>

<TextCell Text="{Binding Name}" // 这里显示名字就行

TextColor="#404040">

</TextCell>

</DataTemplate>

</ListView.ItemTemplate>

</ListView>

点击列表后连接蓝牙

async void btlist\_ItemSelected(System.Object sender, Xamarin.Forms.SelectedItemChangedEventArgs e)

{

if (e.SelectedItem == null) return;

IDevice selecteddevice = e.SelectedItem as IDevice; // 选中的就是Device

((ListView)sender).SelectedItem = null;

Console.WriteLine("seleted is " + selecteddevice.Name);

try

{

await adapter.ConnectToDeviceAsync(selecteddevice); // 连接

Console.WriteLine("Connect to " + selecteddevice.Name + " successfully!");

await DisplayAlert("Successful", "Connect to " + selecteddevice.Name + " successfully!", "Perfect!");

dataviewmodel.record\_device(selecteddevice);

}

catch (DeviceConnectionException)

{

await DisplayAlert("Error", "Cant connect to that device", "OK");

}

}

获取Services – 获取Characteristic – 获取内容

var services = await device.GetServicesAsync();

// services是service的集合，service有自己的ID，有点是没用的，也有有用的，也就是UUID

foreach (var service in services) // 迭代器

{

var characteristics = await service.GetCharacteristicsAsync();

// characteristics是characteristic的集合，每个都有自己的ID，也就是UUID

foreach (var characteristic in characteristics)

{

Console.WriteLine(service.Id + " - " + characteristic.Id);

// 筛选有用的

if (characteristic.Id == Guid.Parse("5bfdc0be-6476-11eb-ae93-0242ac130002"))

{

// 下面这一堆是不断接受信息

characteristic.ValueUpdated += (o, args) =>

{

var bytes = args.Characteristic.Value;

Console.WriteLine(System.Text.Encoding.ASCII.GetString(bytes));

};

await characteristic.StartUpdatesAsync();

}

}

}

# ViewModel

建立一个类，里面包含个个页面需要共享的数据，以及各种方法，例如

namespace TEXT

{

public class DataViewModel

{

public IDevice selectedDevice;

public IBluetoothLE ble;

public IAdapter adapter;

public string DEVICE\_UUID = "ca7b1cb0-144c-40f3-a0e6-409c6bd1a300";

public void record\_device(IDevice device) {}

public IDevice GetDevice() {}

public void record\_BLE(IBluetoothLE bluetooth) {}

public IBluetoothLE GetBLE() {}

public void record\_Adapter(IAdapter ADA) {}

public IAdapter GetAdapter() {}

}

}

在切换页面的时候，把数据都存进去，然后传递到下一个页面

async void Button\_Clicked(object sender, EventArgs e)

{

dataviewmodel.record\_BLE(ble); // 这里在存数据

dataviewmodel.record\_Adapter(adapter);

var detailpage = new DetailPage(dataviewmodel); // 用这种方法产生新的页面，并传入数据

await Navigation.PushModalAsync(detailpage); // 导航到detailpage那一页

}

对应的，在下一页要改一下

public partial class DataReading : ContentPage

{

public DataViewModel dataviewmodel; // 在这里建立DataViewModel和对象

IBluetoothLE ble; // 建立变量

IAdapter adapter;

IDevice device;

public DataReading(DataViewModel m) // 在这里定义传递进去的参数

{

InitializeComponent();

this.dataviewmodel = m; // 绑定上ViewModel

device = dataviewmodel.GetDevice(); // 数据也绑定上

ble = dataviewmodel.GetBLE();

adapter = dataviewmodel.GetAdapter();

}

}

# 字符串分割

public string messagespliter(string message, int v)

{

int comma = message.IndexOf(',');

Console.WriteLine("Got the Message " + message);

if(v == 0)

{

return message.Substring(0, comma);

}

else

{

return message.Substring(comma+1);

}

}

# 与系统交互

获取系统时间

Device.StartTimer(TimeSpan.FromSeconds(0.1), () => // 每0.1秒更新一次

{

year = DateTime.Now.ToString("yyyy");

month = DateTime.Now.ToString("MM");

day = DateTime.Now.ToString("dd");

hour = DateTime.Now.ToString("HH");

minute = DateTime.Now.ToString("mm");

second = DateTime.Now.ToString("ss");

return true; // 一直更新

});

改变字符串"yyyy/MM"之类的内容，可以直接改变输出的形式。

写入系统

# 改变APP名字和图标

Android

在项目.Android/MainActivity.cs改，Label就是APP名字，Icon就是图标，图标可以用png

using System;

// 其他包……

namespace TEXT.Droid

{

[Activity(Label = "AirCase", Icon = "@drawable/AirCase", Theme = "@style/MainTheme"，…

// 其他内容……

}

# C#动态数组

public ArrayList myAL = new ArrayList();

方法

添加

ArrayList.Add()

强制类型转换

int[] array = (int[]) al.ToArray(typeof( int ));