# 本書の想定する開発環境

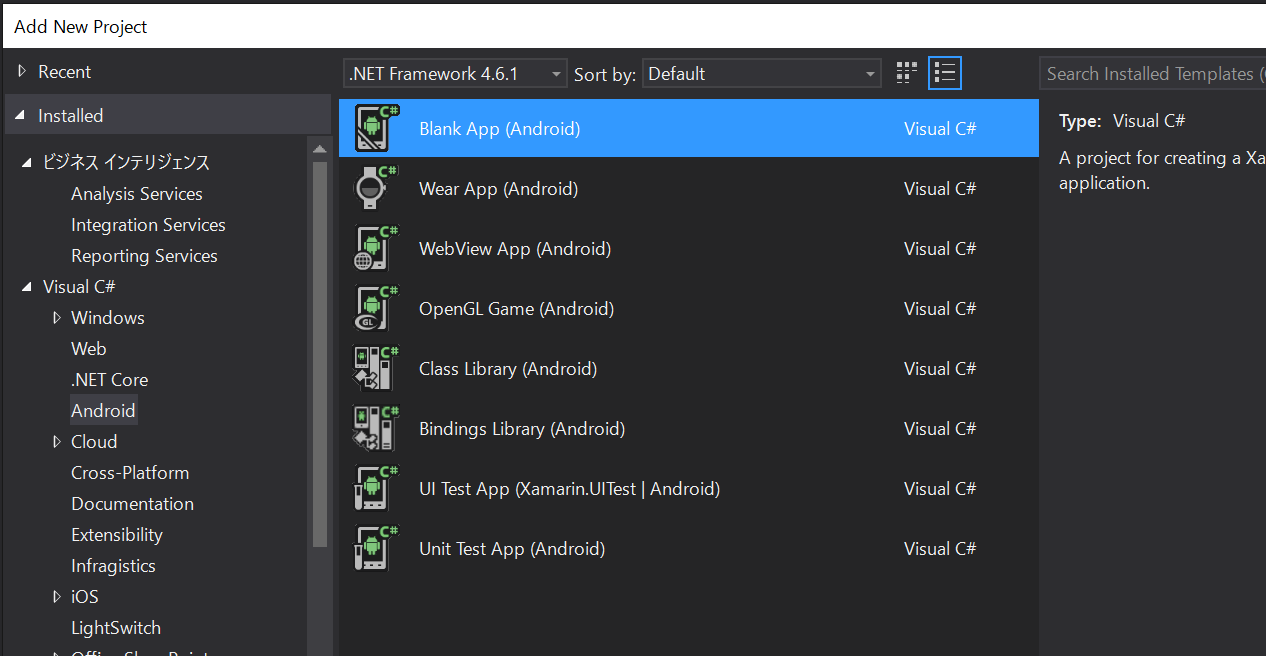
本書では、Xamarin.Androidを使ったアプリケーション開発について解説します。著者も入門しながらなので間違い等あるかもしれませんが、その場合はご指摘ください。以下の開発環境・言語を想定しています。

* Visual Studio 2015（英語の言語パックを当てています）
* Xamarin.Android
* C# 6.0
* Android 6.0（APIレベル23）

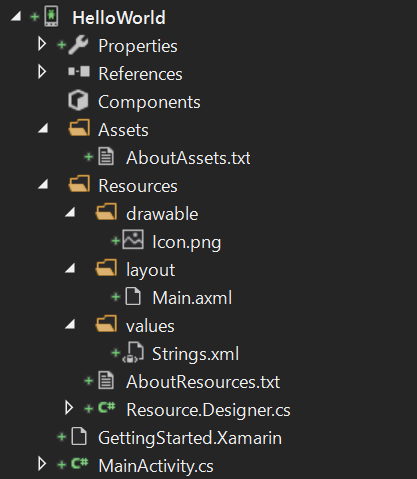
環境構築の方法やIDEの基本的な使い方や、C#についての基本的な文法は知っているものとして解説を行います。これらについて前提知識が十分でないかたは別の書籍やWeb上のサイトなどで情報を得てください。

# 基本的なアプリケーション

基本的なアプリケーションを通じて、簡単なアプリケーションの開発の流れを見てみようと思います。Visual Studioのプロジェクトの新規作成から「Android」→「Blank App（Android）」を選択します。



「HelloWorld」とプロジェクト名をつけてプロジェクトを作成します。そうすると、以下のような構造をもったプロジェクトが作成されます。



MainActivity.csがメインの画面を表すクラスになります。Main.axmlが、MainActivityで使用されている画面のレイアウトを定義したファイルになります。Strings.xmlが、アプリ内で使用する文字列を定義したファイルになります。Resource.Designer.csファイルはMain.axmlやStrings.xmlなどから自動生成される、プログラム内から各種リソースにアクセスするためのIDが定義されたクラスになります。MainActivityクラスは以下のようになっています。

using System;

using Android.App;

using Android.Content;

using Android.Runtime;

using Android.Views;

using Android.Widget;

using Android.OS;

namespace HelloWorld

{

[Activity(Label = "HelloWorld", MainLauncher = true, Icon = "@drawable/icon")]

public class MainActivity : Activity

{

int count = 1;

protected override void OnCreate(Bundle bundle)

{

base.OnCreate(bundle);

// Set our view from the "main" layout resource

SetContentView(Resource.Layout.Main);

// Get our button from the layout resource,

// and attach an event to it

Button button = FindViewById<Button>(Resource.Id.MyButton);

button.Click += delegate { button.Text = string.Format("{0} clicks!", count++); };

}

}

}

まず、ポイントとなるのがActivityクラスを継承している点です。このクラスを継承したものがAndroidでは画面として使えます。次に、このクラスが初期起動時に呼び出されるActivityであることを定義するために属性が定義されています。

[Activity(Label = "HelloWorld", MainLauncher = true, Icon = "@drawable/icon")]

MainLauncher = trueの記述がそれになります。LabelはラベルでIconはアイコンです。Activityが生成されたときにフレームワークから呼び出されるOnCreateメソッドで初期化処理が行われています。SetContentViewメソッドが画面定義ファイルとActivityの紐づけを行うメソッドになります。Resourceクラスが、先ほど説明したResource.Designer.csファイルで自動生成されたファイルでMain.axmlを指し示すためのIDにResource.Layout.Mainでアクセス出来るようになっています。これでMain.axmlで定義された見た目が使用されるようになります。Main.axmlは、以下のように定義されています。

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

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:orientation="vertical"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

<Button

android:id="@+id/MyButton"

android:layout\_width="match\_parent"

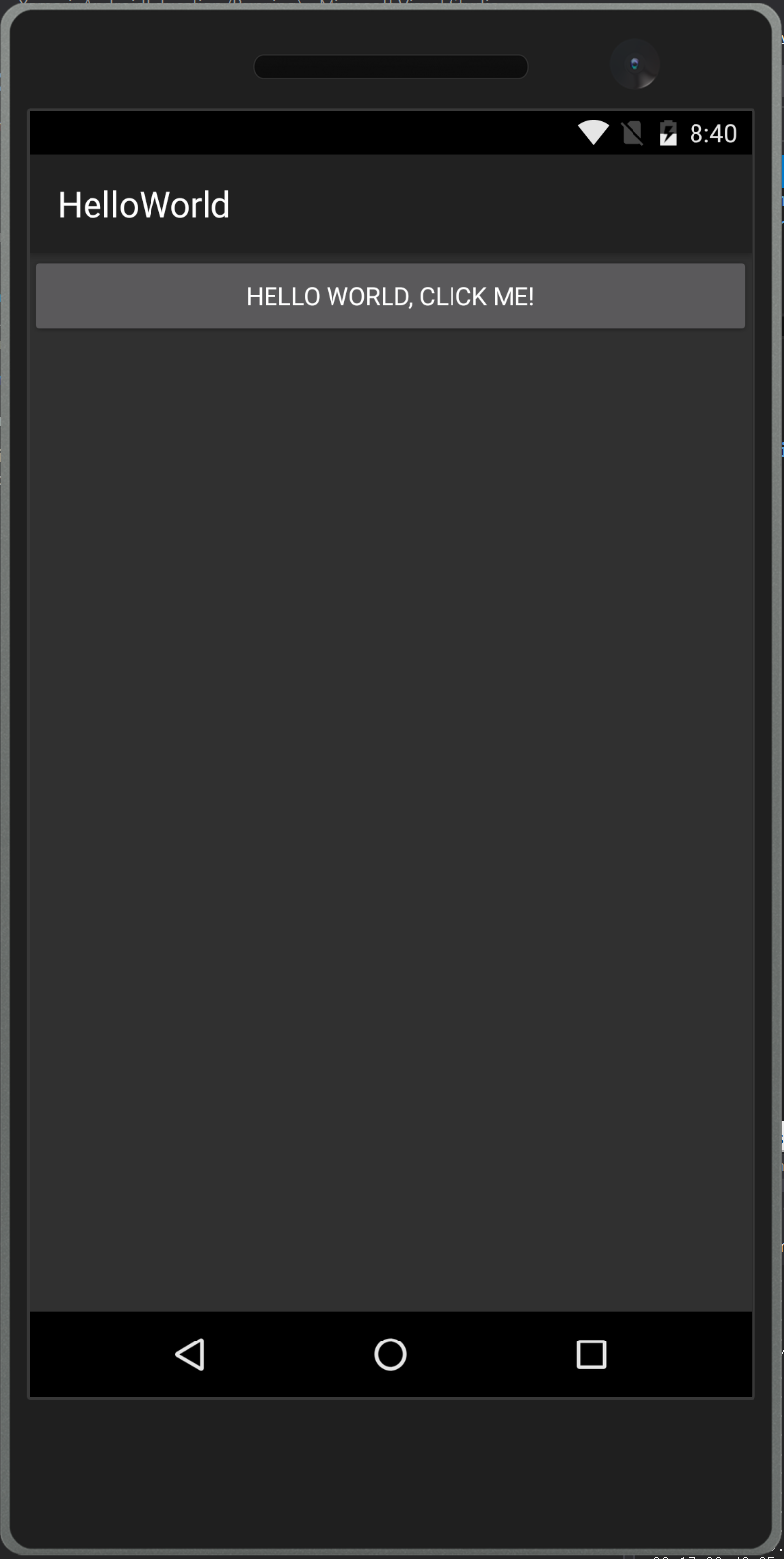
android:layout\_height="wrap\_content"

android:text="@string/Hello" />

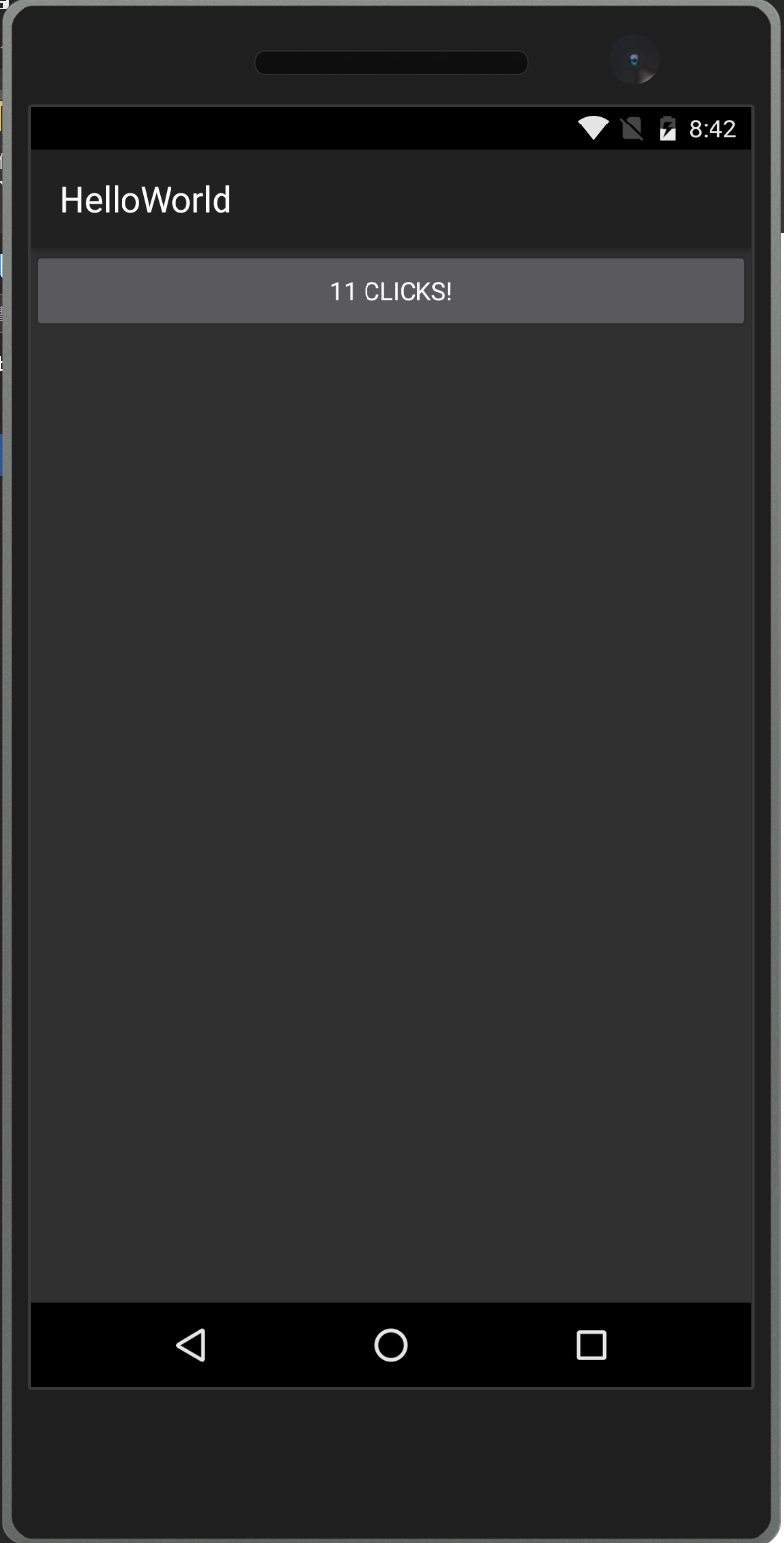
</LinearLayout>

xmlns:android=”http://schemas.android.com/apk/res/android”名前空間が定義されたXMLファイルになります。LinearLayoutという要素を縦や横に並べるレイアウトの中に、Buttonが置かれているというくらいの理解で今は大丈夫（というかこれを書いてる時点で、その程度の理解です）だと思います。ポイントとして、押さえておきたいのが、Buttonに対して定義されているandroid:id=”@+id/MyButton”という属性です。これはMyButtonという名前のIDをボタンに割り当ててることになります。

ではMainActivityに戻ります。MainActivityでは、FindViewByIdメソッドを使って画面内のコントロールにアクセス出来ます。ここでは、MyButtonというIDのコントロール（先ほど確認したMain.axml内で定義されてるやつですね）を取得してイベントを登録しています。処理自体はcount変数をカウントアップしているだけのシンプルなものですね。では、実行してみましょう。



ボタンを押すとカウントアップされた値が表示されます。



## BMIを計算してみよう

これを少しいじってBMIを計算するアプリを作ってみようと思います。身長と体重を入力するボックスを作ります。Main.axmlを以下のように変更します。

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

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:orientation="vertical"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

<Button

android:id="@+id/MyButton"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:text="@string/Hello" />

<EditText

android:id="@+id/EditTextHeight"

android:hint="@string/Height"

android:inputType="numberDecimal"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content" />

<EditText

android:id="@+id/EditTextWeight"

android:hint="@string/Weight"

android:inputType="numberDecimal"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content" />

</LinearLayout>

EditTextがAndroidのテキスト入力用コントロールになります。android:hint属性は、ウォーターマークをだすための属性になります。@string/Heightや@string/Weightは、strings.xmlで以下のように定義している文字列を参照する書き方です。BmiMessageは、プログラム内から使用する予定の文字列になります。

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

<resources>

<string name="Hello">Hello World, Click Me!</string>

<string name="ApplicationName">HelloWorld</string>

<string name="Weight">体重</string>

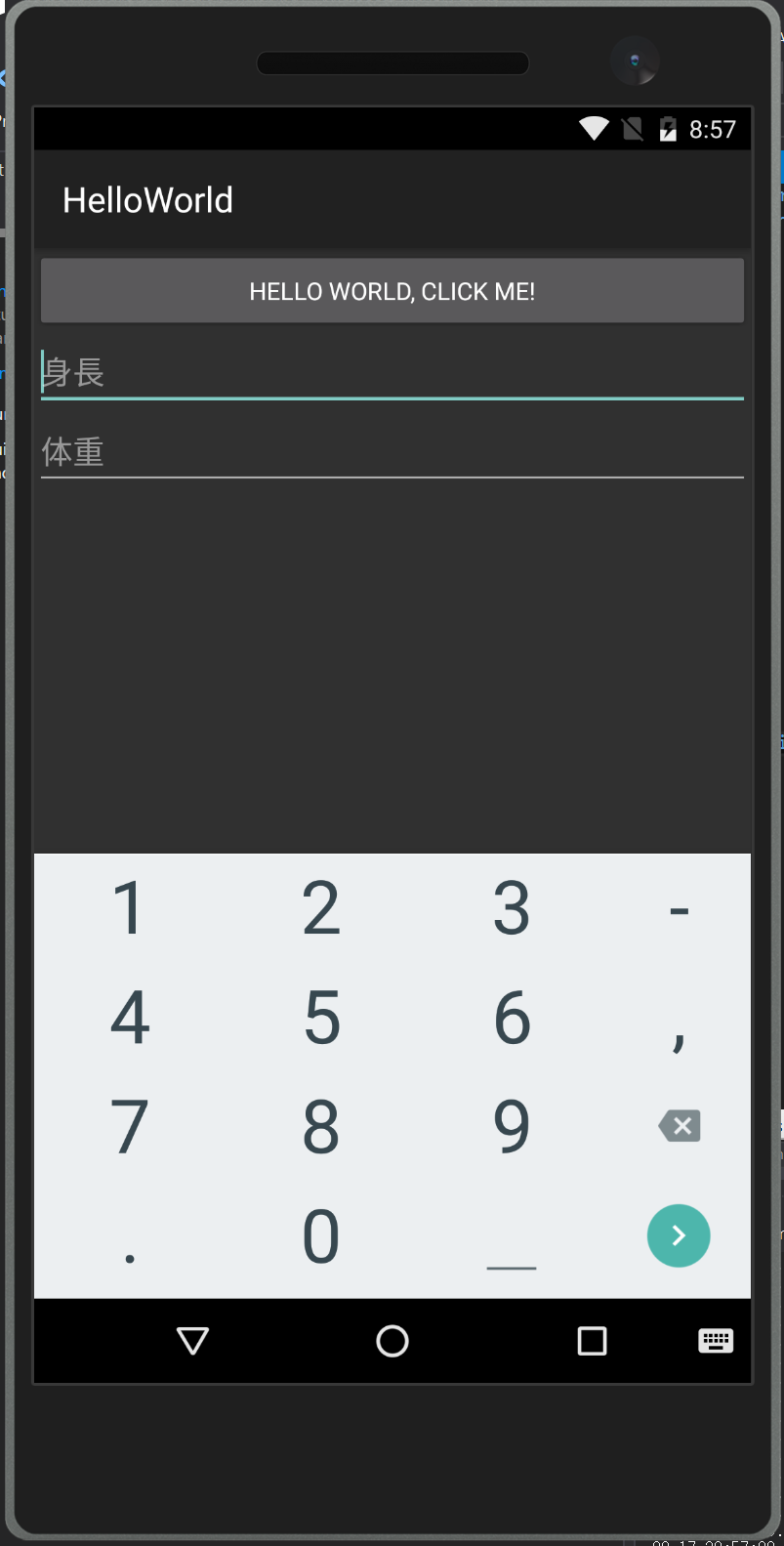
<string name="Height">身長</string>

<string name="BmiMessage">BMIは%fです</string>

</resources>

android:inputType=”numberDecimal”は、数値を入力するための設定になります。android:layout\_widthとandorid:layout\_heightは、幅と高さをどうするかを指定します。今回は横幅は親に合わせて、縦幅はコンテンツに合わせるように指定しています。（つまり横長）

実行すると、以下のように表示されます。



身長、体重のウォーターマークが出ている点と、入力欄にフォーカスを合わせたときのソフトウェアキーボードが数字になっているので、先ほどの設定が効いていることがわかります。

最後に、処理を書いていきます。BMIの計算式は「体重(kg) ÷ {身長(m) Ｘ 身長(m)}」なので、ボタンが押されたときに、そう表示されるようにしています。アプリの仕様的にわかりにくいですが、今回は体重はkgで、身長はmで入力されてるものとして処理しています。

using Android.App;

using Android.OS;

using Android.Widget;

namespace HelloWorld

{

[Activity(Label = "HelloWorld", MainLauncher = true, Icon = "@drawable/icon")]

public class MainActivity : Activity

{

protected override void OnCreate(Bundle bundle)

{

base.OnCreate(bundle);

this.SetContentView(Resource.Layout.Main);

var button = this.FindViewById<Button>(Resource.Id.MyButton);

var height = this.FindViewById<EditText>(Resource.Id.EditTextHeight);

var weight = this.FindViewById<EditText>(Resource.Id.EditTextWeight);

button.Click += (\_, \_\_) =>

{

double h;

if (!double.TryParse(height.Text, out h))

{

return;

}

double w;

if (!double.TryParse(weight.Text, out w))

{

return;

}

// 体重(kg) ÷ {身長(m) Ｘ 身長(m)}

var bmi = w / (h \* h);

button.Text = string.Format(this.Resources.GetString(Resource.String.BmiMessage ,bmi));

};

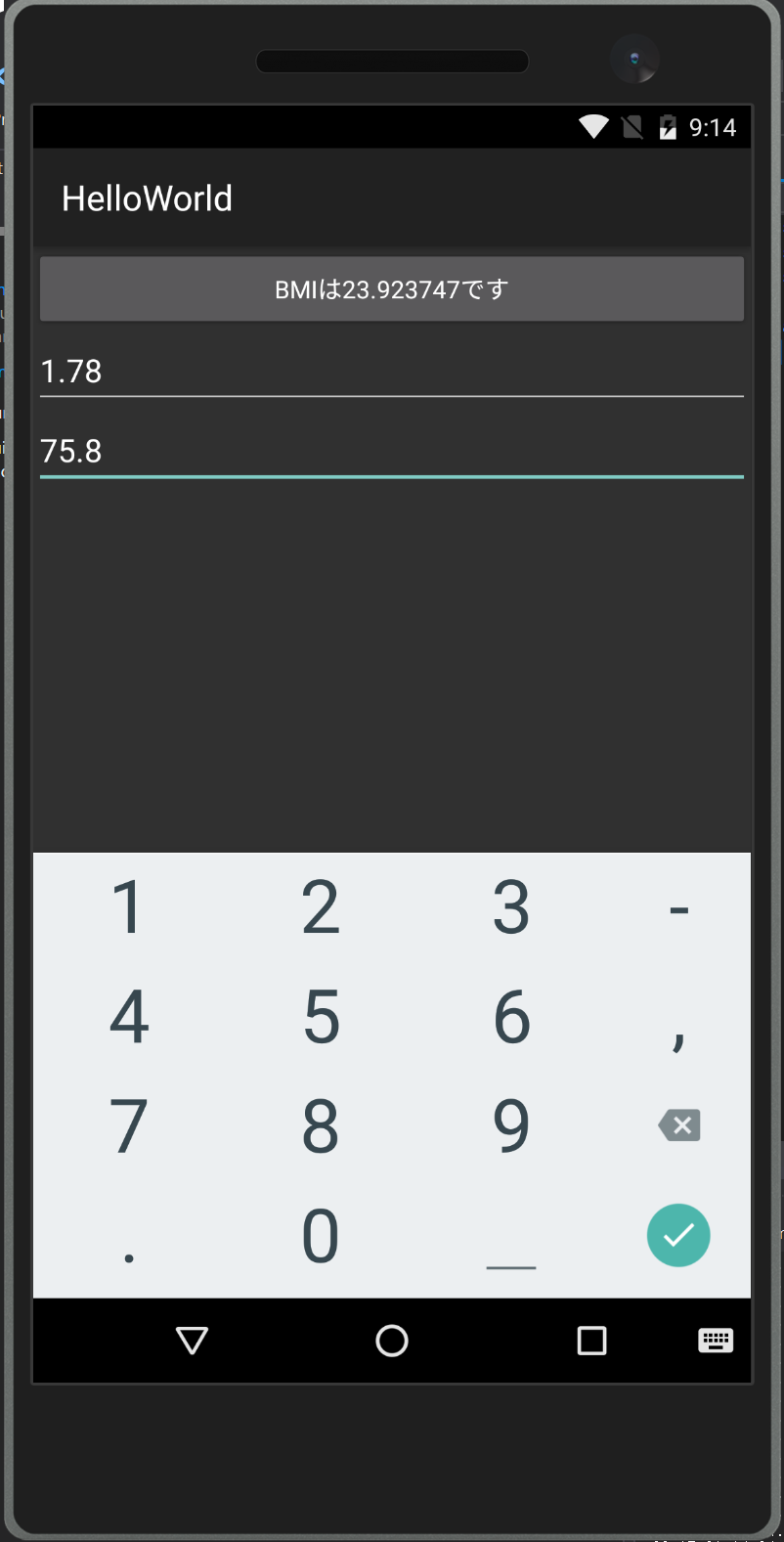
}

}

}

ActivityにResourcesというプロパティがあり、これのGetStringメソッドを使うことでStrings.xmlに定義した文字列をプログラム中から使うことが出来ます。今回はBMIの計算結果を表示する文字列として使っています。

実行して入力してボタンを押すと以下のようになります。



# 画面遷移をしてみよう

ここでは、Andoridの画面遷移について説明します。Andoridでは、画面遷移にIntentというものを使います。このIntentは、とても汎用的なメッセージング機構でサービスとよばれるバックグラウンドで実行される処理の起動や、ここで説明する別画面（Activity）の起動などができます。さらには、別アプリのActivityやサービスなども起動することもできます。Intentは、簡単に言うと宛先とデータを持った入れ物です。それを投げつけると、それに応答するように設定されたものが応答します。

では、画面遷移するアプリケーションを作ってIntentの簡単な使い方を見てみたいと思います。NavigationAppという名前でAndroidのBlank Appを作成します。そして、新規作成から、ActivityをNextActivityという名前で作成します。NextActivityに対応する見た目を定義するaxmlをResources/layoutフォルダにNext.axmlという名前で作成します。

Main.axmlにEditTextを追加します。idは@+id/GreetMessageにしました。axmlを以下に示します。

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

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:orientation="vertical"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:minWidth="25px"

android:minHeight="25px">

<EditText

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:id="@+id/GreetMessage" />

</LinearLayout>

## メニューの定義

ここで、画面遷移のほかに新しいことをやってみたいと思います。メニューをアクションバーに追加してみたいと思います。メニューは、ActivityのOnCreateOptionsMenuメソッドをオーバーライドしてMenuInflaterのInflateメソッドを使うことで作成します。メニューは、Resources/menuフォルダにXMLで定義します。ここでは、MainMenu.xmlという名前で以下のようなXMLを定義しました。

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

<menu xmlns:android="http://schemas.android.com/apk/res/android">

<item android:id="@+id/NavigateMenu"

android:title="@string/NavigateMenuText"

android:onClick="NavigateMenuClick"

android:showAsAction="always"

android:icon="@drawable/Icon"/>

</menu>

android:idに識別用のIDを定義し、android:titleにメニューを長押ししたときに表示されるテキストを指定し、android:onClickに選択時に実行されるメソッドを指定し、android:showAsActionにアクションバーへの表示方法を指定し、android:iconにアイコンを指定します。android:showAsActionには、バーには表示しないnever、余裕があればバーに表示するifRoom、常に表示するalways、android:titleのテキストを表示するwithTextなどがあります。その他の完全な属性の定義については、以下のページを参照してください。

<https://developer.android.com/guide/topics/resources/menu-resource.html>

itemは複数定義することが出来ます。また、menu/menu/itemのようにメニューにメニューを入れ子にすることもできます。

上記メニューを画面に表示します。MainActivityのOnCreateOptionsMenuでMenuInflaterのInflateメソッドでメニューのリソースと引数で渡されたIMenuを結びつけます。そして、NavigateMenuClickメソッドを定義して選択時のアクションを定義します。NavigateMenuClickは、Java側にメソッドを教えるためにMono.Android.dllアセンブリ（追加で参照が必要）に定義されているJava.Interop.Export属性で名前を指定します。コードは以下のようになります。

using System;

using Android.App;

using Android.Content;

using Android.Runtime;

using Android.Views;

using Android.Widget;

using Android.OS;

using Android.Util;

using Java.Interop;

namespace NavigationApp

{

[Activity(Label = "NavigationApp", MainLauncher = true, Icon = "@drawable/icon")]

public class MainActivity : Activity

{

protected override void OnCreate(Bundle bundle)

{

base.OnCreate(bundle);

// Set our view from the "main" layout resource

this.SetContentView(Resource.Layout.Main);

}

public override bool OnCreateOptionsMenu(IMenu menu)

{

this.MenuInflater.Inflate(Resource.Menu.MainMenu, menu);

return true;

}

[Export(nameof(NavigateMenuClick))]

public void NavigateMenuClick(IMenuItem menuItem)

{

Log.Debug("MainActivity", $"Clicked: {menuItem.TitleFormatted}");

}

}

}

これで、実行すると以下のような画面が表示され、アクションバーのメニューを選択することで出力ウィンドウにメッセージが表示されます。



## 画面遷移

メニューを選択したときに画面遷移を行うようにします。以下のようにNextActivityを対象としたIntentを作成して、PutExtraメソッドでIntentにデータを設定してStartActivityメソッドで次の画面を起動します。

[Export(nameof(NavigateMenuClick))]

public void NavigateMenuClick(IMenuItem menuItem)

{

// NextActivityへのIntentを作成して

var intent = new Intent(this, typeof(NextActivity));

// データを詰めて

intent.PutExtra("Message", this.FindViewById<EditText>(Resource.Id.GreetMessage).Text);

this.StartActivity(intent);

}

NextActivity用の画面であるNext.axmlではTextViewを1つおいて受け取ったメッセージを表示したいと思います。axmlの内容を以下に示します。

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

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:orientation="vertical"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:minWidth="25px"

android:minHeight="25px">

<TextView

android:text="Text"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:id="@+id/TextViewGreetMessage" />

</LinearLayout>

画面遷移先のNextActivityでは、Intentプロパティを通じて渡されたIntentが参照できます。ここからGetStringExtraメソッドで渡された値を参照してTextViewに設定しています。

using Android.App;

using Android.OS;

using Android.Widget;

namespace NavigationApp

{

[Activity(Label = "NextActivity")]

public class NextActivity : Activity

{

protected override void OnCreate(Bundle savedInstanceState)

{

base.OnCreate(savedInstanceState);

this.SetContentView(Resource.Layout.Next);

var textView = this.FindViewById<TextView>(Resource.Id.TextViewGreetMessage);

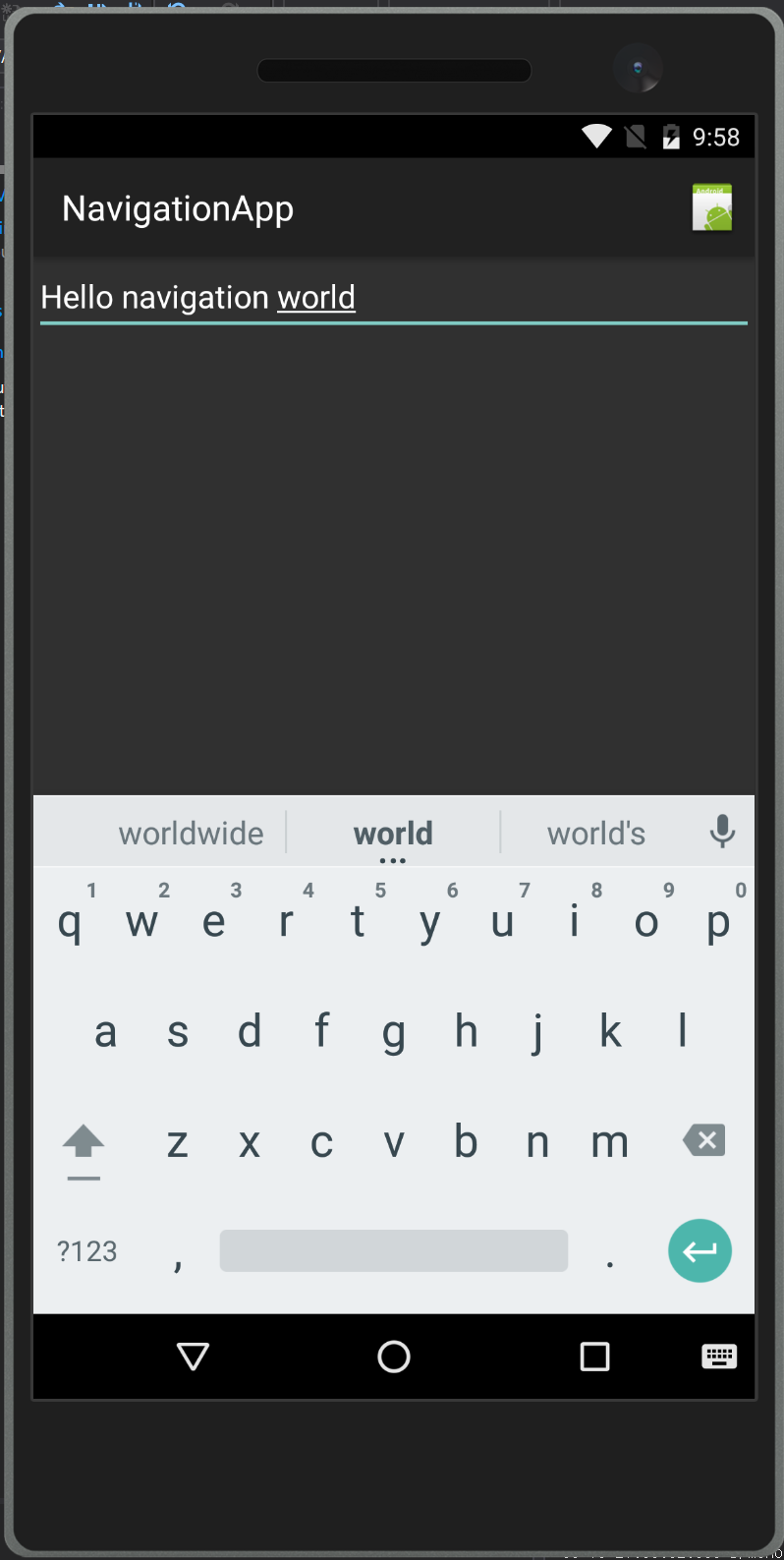
textView.Text = this.Intent.GetStringExtra("Message");

}

}

}

実行すると、以下のようになります。





画面遷移が行われて、値が渡されていることが確認できます。

## Activityのライフサイクル

Activityのライフサイクルについて説明します。Activityのライフサイクルで呼び出されるメソッドは以下の7つがあります。があります。

* OnCreateメソッド  
  Activityが作成されたときに呼び出される。
* OnStartメソッド  
  アクティビティがユーザーに表示される直前に呼び出される。
* OnResumeメソッド  
  アクティビティがユーザーとの操作が開始される直前に呼び出される。
* OnPauseメソッド  
  別のアクティビティを表示する直前に呼び出される
* OnStopメソッド  
  アクティビティがユーザーから見えなくなると表示される
* OnDestroyメソッド  
  アクティビティが破棄される前に呼び出される
* OnRestartメソッド  
  アクティビティが停止したあと再開する直前に呼び出される

通常は、OnCreateメソッドで初期化処理を行い、OnPauseメソッドで未保存の永続化データを保存すると良いでしょう。別のActivityが表示されると、もともと表示されていたActivityは、通常OnPause→OnStopが呼び出されて次の表示が来るまで待ちますが、メモリが圧迫されたりするとActivityが破棄されたりします。そうなると、次に戻ってきたときはOnCreateからやり直しになります。その時、EditTextなどの入力途中のデータなどは、IDが振られていると、自動的に復元されます。その他のユーザーがActivityのフィールドなどに保持していたデータは何もしないとクリアされてしまいます。これに対応するためには、OnSaveInstanceStateメソッドをオーバーライドして、Bundleにデータを保存します。OnCreateの引数で渡されるBundleがnullじゃないときは、保存されたデータがあるということなので、データの復元をBundleから行います。

この動作を確認するためのプログラムを以下に示します。MainActivityとNextActivityを持っただけのシンプルなプロジェクトで、以下のようなコードを記述します。

using Android.App;

using Android.Content;

using Android.OS;

using Android.Util;

using Android.Views;

using Java.Interop;

using System;

namespace ActivityLifecycle

{

[Activity(Label = "ActivityLifecycle", MainLauncher = true, Icon = "@drawable/icon")]

public class MainActivity : Activity

{

private string Id { get; set; }

protected override void OnSaveInstanceState(Bundle outState)

{

base.OnSaveInstanceState(outState);

Log.Debug(nameof(MainActivity), $"{nameof(OnSaveInstanceState)}: {this.Id}");

outState.PutString(nameof(Id), this.Id);

}

protected override void OnCreate(Bundle bundle)

{

base.OnCreate(bundle);

// Set our view from the "main" layout resource

this.SetContentView(Resource.Layout.Main);

if (bundle == null)

{

this.Id = Guid.NewGuid().ToString();

}

else

{

this.Id = bundle.GetString(nameof(Id));

}

Log.Debug(nameof(MainActivity), $"{nameof(OnCreate)}: {this.Id}");

}

protected override void OnStart()

{

base.OnStart();

Log.Debug(nameof(MainActivity), $"{nameof(OnStart)}: {this.Id}");

}

protected override void OnResume()

{

base.OnResume();

Log.Debug(nameof(MainActivity), $"{nameof(OnResume)}: {this.Id}");

}

protected override void OnRestart()

{

base.OnRestart();

Log.Debug(nameof(MainActivity), $"{nameof(OnRestart)}: {this.Id}");

}

protected override void OnPause()

{

base.OnPause();

Log.Debug(nameof(MainActivity), $"{nameof(OnPause)}: {this.Id}");

}

protected override void OnStop()

{

base.OnStop();

Log.Debug(nameof(MainActivity), $"{nameof(OnStop)}: {this.Id}");

}

protected override void OnDestroy()

{

base.OnDestroy();

Log.Debug(nameof(MainActivity), $"{nameof(OnDestroy)}: {this.Id}");

}

[Export(nameof(MyButtonClick))]

public void MyButtonClick(View v)

{

this.StartActivity(new Intent(this, typeof(NextActivity)));

}

}

}

ポイントは、OnSaveInstanceStateメソッドとOnCreateメソッドになります。それ以外はボタンが押されたときの画面遷移の処理と、各ライフサイクルメソッドが呼び出されたかログを出力するためのものになります。アプリケーションを実行してMainActivityを表示するとログは以下のようなものが表示されます。

08-20 15:36:59.111 D/MainActivity( 2702): OnCreate: 6986edf6-5759-46a9-98be-4f9185ad23de

08-20 15:36:59.116 D/MainActivity( 2702): OnStart: 6986edf6-5759-46a9-98be-4f9185ad23de

08-20 15:36:59.117 D/MainActivity( 2702): OnResume: 6986edf6-5759-46a9-98be-4f9185ad23de

そして、NextActivityへ遷移すると以下のように表示されます。

08-20 15:37:51.231 D/MainActivity( 2702): OnPause: 6986edf6-5759-46a9-98be-4f9185ad23de

08-20 15:37:52.000 D/MainActivity( 2702): OnSaveInstanceState: 6986edf6-5759-46a9-98be-4f9185ad23de

08-20 15:37:52.001 D/MainActivity( 2702): OnStop: 6986edf6-5759-46a9-98be-4f9185ad23de

そして、戻るボタンでMainActivityに戻ると以下のように表示されます。

08-20 15:39:41.493 D/MainActivity( 2702): OnRestart: 6986edf6-5759-46a9-98be-4f9185ad23de

08-20 15:39:41.493 D/MainActivity( 2702): OnStart: 6986edf6-5759-46a9-98be-4f9185ad23de

08-20 15:39:41.493 D/MainActivity( 2702): OnResume: 6986edf6-5759-46a9-98be-4f9185ad23de

OnDestroyメソッドが呼び出されていないので、Idは保持されたままなのは当然ですよね。次に、実機の開発者オプションで「アクティビティを保持しない」を選択して、疑似的にメモリ不足などでActivityが破棄された時の動作をエミュレートしてみたいと思います。実行してMainActivityが表示されると以下のようなログが出ます。

08-20 15:42:38.221 D/MainActivity(31579): OnCreate: dbc9cedb-a67a-4e61-b6b1-911925fbd001

08-20 15:42:38.223 D/MainActivity(31579): OnStart: dbc9cedb-a67a-4e61-b6b1-911925fbd001

08-20 15:42:38.225 D/MainActivity(31579): OnResume: dbc9cedb-a67a-4e61-b6b1-911925fbd001

そして、NextActivityへ遷移を行います。

08-20 15:43:10.217 D/MainActivity(31579): OnPause: dbc9cedb-a67a-4e61-b6b1-911925fbd001

08-20 15:43:10.667 D/MainActivity(31579): OnSaveInstanceState: dbc9cedb-a67a-4e61-b6b1-911925fbd001

08-20 15:43:10.673 D/MainActivity(31579): OnStop: dbc9cedb-a67a-4e61-b6b1-911925fbd001

08-20 15:43:10.689 D/MainActivity(31579): OnDestroy: dbc9cedb-a67a-4e61-b6b1-911925fbd001

OnDestroyメソッドが呼び出されてMainActivityが破棄されたことが確認できます。この状態で戻るボタンを押してMainActivityに戻ると以下のように表示されます。

08-20 15:44:12.607 D/MainActivity(31579): OnCreate: dbc9cedb-a67a-4e61-b6b1-911925fbd001

08-20 15:44:12.609 D/MainActivity(31579): OnStart: dbc9cedb-a67a-4e61-b6b1-911925fbd001

08-20 15:44:12.624 D/MainActivity(31579): OnResume: dbc9cedb-a67a-4e61-b6b1-911925fbd001

一度MainActivityが破棄されたにも関わらず、Idの値が保持されていることが確認できます。繰り返しますが、画面に置かれた部品でIDが振られたものは自動的に値が保持されるため気にする必要はありませんが、今回のように自分でActivityに一時的に保持しているものは自前で保存と復元を行う必要があります。