Day 13: Controls: Input Controls, Part 2

In today’s post we will go over few of the rest of the input controls in Xamarin.Android

# Switch

A Switch control can toggle between two states, ON or OFF. The Switch control’s default is value is OFF. The text that is shown for the Switch, it’s ON and OFF text and its default state are all configurable using AXML in Xamarin Android.

Let’s look at a simple Switch control AXML –

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  android:orientation="vertical"  android:layout\_width="fill\_parent"  android:layout\_height="fill\_parent">  <Switch  android:text="Send Error Logs to Developer?"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:checked="true"  android:textOn="YES"  android:textOff="NO"  android:id="@+id/switch1" />  </LinearLayout> |

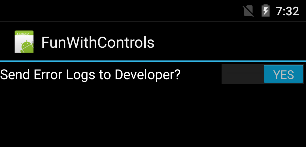
Gist file link: <https://gist.github.com/vkoppaka/97236a1aff7585e55b16>

## Properties

In the XML snippet above there are 4 properties that are important to a Switch control –

* **android:text:** This property sets the text that is shown before Switch control.
* **android:checked:** This property controls the default state of the Switch control. Valid values for this property are “true” or “false”
* **android:textOn:** This property controls the text that is shown when the Switch state is ON.
* **android:textOff:** This property control the text that is shown when the Switch state is OFF.

If you run the app with the above Switch, the UI would look like –



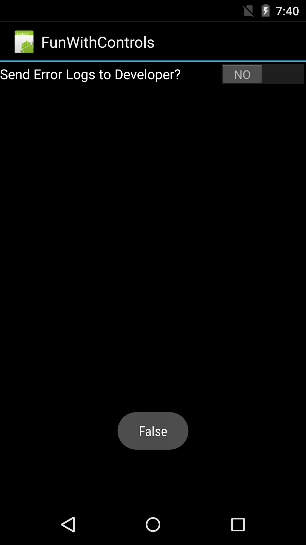
## Events:

The most important event that Switch Control exposes is the CheckedChange event which fires when Switch Controls’ state changes between ON and OFF. Now let us see how to subscribe to this event in code –

|  |
| --- |
| using Android.App;  using Android.OS;  using Android.Widget;  namespace FunWithControls  {  [Activity(Label = "FunWithControls", 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  SetContentView(Resource.Layout.Main);  var switchView = FindViewById<Switch>(Resource.Id.switch1);  switchView.CheckedChange += switchView\_CheckedChange;  }  void switchView\_CheckedChange(object sender, CompoundButton.CheckedChangeEventArgs e)  {  Toast.MakeText(this, e.IsChecked.ToString(), ToastLength.Long).Show();  }  }  } |

Gist file link: <https://gist.github.com/vkoppaka/f7cf3eece0946db2af74>

The CheckedChange event passes down a CheckedChangeEventArgs which has a property to indicate whether the current state is Checked or not.



# CheckBox

A checkbox is also a control that can toggle between two states: Checked and UnChecked. CheckBox control has been available in Android API since the very first days, while Switch Control was introduced to the Android API Level 14. Similar to Switch, a CheckBox’s default state is OFF, or rather, UnChecked. You can customize the text that is shown with Checkbox and also control the state of the checkbox using the properties. Let’s see a Simple Checkbox in action –

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  android:orientation="vertical"  android:layout\_width="fill\_parent"  android:layout\_height="fill\_parent"  android:minWidth="25px"  android:minHeight="25px">  <CheckBox  android:text="Include this item in download?"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:id="@+id/checkBox" />  </LinearLayout> |

Gist file link: <https://gist.github.com/vkoppaka/37d4c60c7bbdfe74e215>

## Properties

The two most important properties for Checkbox are –

* **android:text:** This property sets the text that is shown after the checkbox control.
* **android:checked:** This property controls the default state of the checkbox control. Valid values for this property are “true” or “false”

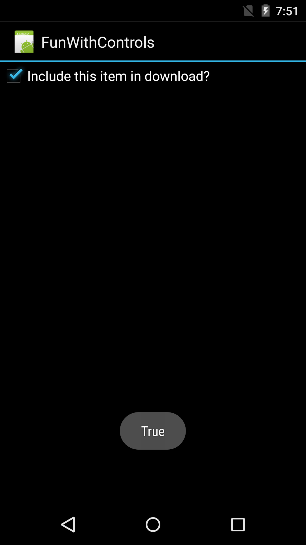
## Events

The most important event that CheckBox Control exposes is the CheckedChange event which fires when CheckBox Controls’ state changes between Checked and UnChecked. Now let us see how to subscribe to this event in code –

|  |
| --- |
| using Android.App;  using Android.OS;  using Android.Widget;  namespace FunWithControls  {  [Activity(Label = "FunWithControls", MainLauncher = true, Icon = "@drawable/icon")]  public class CheckBoxSampleActivity : Activity  {  protected override void OnCreate(Bundle bundle)  {  base.OnCreate(bundle);  // Set our view from the "main" layout resource  SetContentView(Resource.Layout.CheckBoxSample);  var checkBox = FindViewById<CheckBox>(Resource.Id.checkBox);  checkBox.CheckedChange += checkBox\_CheckedChange;  Toast.MakeText(this, checkBox.Checked.ToString(), ToastLength.Long).Show();  }  void checkBox\_CheckedChange(object sender, CompoundButton.CheckedChangeEventArgs e)  {  Toast.MakeText(this, e.IsChecked.ToString(), ToastLength.Long).Show();  }  }  } |

Gist file link: <https://gist.github.com/vkoppaka/a2b4a1b8f4b6c32b8f38>

The CheckedChange event passes down a CheckedChangeEventArgs which has a property to indicate whether the current state is Checked or not.



# RadioButton

A RadioButton is also a control that can toggle between two states: Checked and UnChecked. RadioButton’s default state is OFF, or rather, UnChecked. You can customize the text that is shown with RadioButton and also control the state of the RadioButton using the properties. Let’s see a Simple RadioButton in action –

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  android:orientation="vertical"  android:layout\_width="fill\_parent"  android:layout\_height="fill\_parent"  android:minWidth="25px"  android:minHeight="25px">  <RadioButton  android:text="This is a simple Radio Button"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:id="@+id/radioButton" />  </LinearLayout> |

Gist file link: <https://gist.github.com/vkoppaka/8c3ca586407ad0bf82c7>

## Properties

The two most important properties for RadioButton are –

* **android:text:** This property sets the text that is shown after the RadioButton control.
* **android:checked:** This property controls the default state of the RadioButton control. Valid values for this property are “true” or “false”

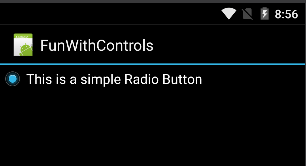
## Events

The most important event that RadioButton Control exposes is the CheckedChange event which fires when RadioButton Controls’ state changes between Checked and UnChecked. Now let us see how to subscribe to this event in code –

|  |
| --- |
| using Android.App;  using Android.OS;  using Android.Widget;  namespace FunWithControls  {  [Activity(Label = "FunWithControls", MainLauncher = true, Icon = "@drawable/icon")]  public class RadioButtonSampleActivity : Activity  {  protected override void OnCreate(Bundle bundle)  {  base.OnCreate(bundle);  // Set our view from the "main" layout resource  SetContentView(Resource.Layout.RadioButtonSample);  var radioButton = FindViewById<RadioButton>(Resource.Id.radioButton);  Toast.MakeText(this, radioButton.Checked.ToString(), ToastLength.Long).Show();  radioButton.CheckedChange += radioButton\_CheckedChange;  }  void radioButton\_CheckedChange(object sender, CompoundButton.CheckedChangeEventArgs e)  {  Toast.MakeText(this, e.IsChecked.ToString(), ToastLength.Long).Show();  }  }  } |

Gist file link: <https://gist.github.com/vkoppaka/a632eef38a6498d2ce1d>

The CheckedChange event passes down a CheckedChangeEventArgs which has a property to indicate whether the current state is Checked or not.



# RadioButtonGroup

RadioButtonGroup control is a group of RadioButtons that are mutually exclusive to each other. Let’s see how to define a RadioButtonGroup View –

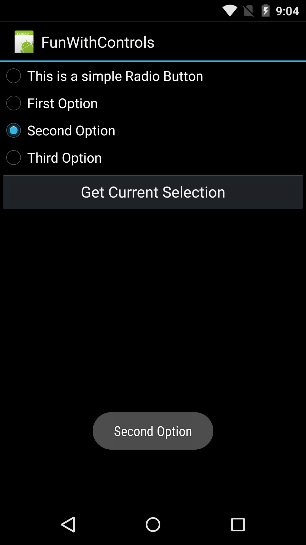
|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  android:orientation="vertical"  android:layout\_width="fill\_parent"  android:layout\_height="fill\_parent"  android:minWidth="25px"  android:minHeight="25px">  <RadioButton  android:text="This is a simple Radio Button"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:id="@+id/radioButton" />  <RadioGroup  android:minWidth="25px"  android:minHeight="25px"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:id="@+id/myRadioGroup">  <RadioButton  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:checked="true"  android:text="First Option"  android:id="@+id/radioButton1" />  <RadioButton  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:text="Second Option"  android:id="@+id/radioButton2" />  <RadioButton  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:text="Third Option"  android:id="@+id/radioButton3" />  </RadioGroup>  <Button  android:text="Get Current Selection"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:id="@+id/button1" />  </LinearLayout> |

Gist file link: <https://gist.github.com/vkoppaka/c30d8138d1c686e9e038>

And in the activity code, to get the currently selected radio button in the RadioButtonGroup we can call CheckedRadioButtonId

|  |
| --- |
| using Android.App;  using Android.OS;  using Android.Widget;  namespace FunWithControls  {  [Activity(Label = "FunWithControls", MainLauncher = true, Icon = "@drawable/icon")]  public class RadioButtonSampleActivity : Activity  {  protected override void OnCreate(Bundle bundle)  {  base.OnCreate(bundle);  // Set our view from the "main" layout resource  SetContentView(Resource.Layout.RadioButtonSample);  var radioButton = FindViewById<RadioButton>(Resource.Id.radioButton);  Toast.MakeText(this, radioButton.Checked.ToString(), ToastLength.Long).Show();  radioButton.CheckedChange += radioButton\_CheckedChange;  var button = FindViewById<Button>(Resource.Id.button1);  button.Click += button\_Click;  }  void button\_Click(object sender, System.EventArgs e)  {  var radioButtonGroup = FindViewById<RadioGroup>(Resource.Id.myRadioGroup);  int selectedRadioButtonId = radioButtonGroup.CheckedRadioButtonId;  var selectedRadioButton = FindViewById<RadioButton>(selectedRadioButtonId);  Toast.MakeText(this, selectedRadioButton.Text, ToastLength.Long).Show();  }  void radioButton\_CheckedChange(object sender, CompoundButton.CheckedChangeEventArgs e)  {  Toast.MakeText(this, e.IsChecked.ToString(), ToastLength.Long).Show();  }  }  } |

Gist file link: <https://gist.github.com/vkoppaka/78a0be4099affd5ce224>

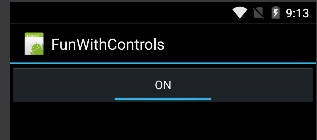


# ToggleButton

ToggleButton is very simple to other two-state Views we have looked at so far. It can exist in two states, track the CheckChanged event and set properties similar to other Input Controls –

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  android:orientation="vertical"  android:layout\_width="fill\_parent"  android:layout\_height="fill\_parent"  android:minWidth="25px"  android:minHeight="25px">  <ToggleButton  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:id="@+id/myToggleButton" />  </LinearLayout> |

|  |
| --- |
| using Android.App;  using Android.OS;  using Android.Widget;  namespace FunWithControls  {  [Activity(Label = "FunWithControls", MainLauncher = true, Icon = "@drawable/icon")]  public class ToggleSampleActivity : Activity  {  protected override void OnCreate(Bundle bundle)  {  base.OnCreate(bundle);  // Set our view from the "main" layout resource  SetContentView(Resource.Layout.ToggleSample);  var toggleButton = FindViewById<ToggleButton>(Resource.Id.myToggleButton);  Toast.MakeText(this, toggleButton.Checked.ToString(), ToastLength.Long).Show();  toggleButton.CheckedChange += toggleButton\_CheckedChange;  }  void toggleButton\_CheckedChange(object sender, CompoundButton.CheckedChangeEventArgs e)  {  Toast.MakeText(this, e.IsChecked.ToString(), ToastLength.Long).Show();  }  }  } |



# RatingBar

RatingBar is an Android View which is used to show rating of any particular thing in starts. Let’s take a look at a simple RatingBar control in Xamarin Andriod –

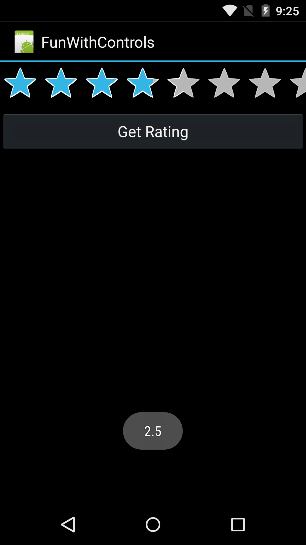
|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  android:orientation="vertical"  android:layout\_width="fill\_parent"  android:layout\_height="fill\_parent"  android:minWidth="25px"  android:minHeight="25px">  <RatingBar  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:id="@+id/myRatingBar" />  </LinearLayout> |

Gist file link: <https://gist.github.com/vkoppaka/245823c9e84eff583695>

And to get a Rating on the RatingBar control, we should call “Rating” property –

|  |
| --- |
| using System.Globalization;  using Android.App;  using Android.OS;  using Android.Widget;  namespace FunWithControls  {  [Activity(Label = "FunWithControls", MainLauncher = true, Icon = "@drawable/icon")]  public class RatingSampleActivity : Activity  {  protected override void OnCreate(Bundle bundle)  {  base.OnCreate(bundle);  // Set our view from the "main" layout resource  SetContentView(Resource.Layout.RatingSample);  var button = FindViewById<Button>(Resource.Id.getRatingButton);  button.Click += button\_Click;  }  void button\_Click(object sender, System.EventArgs e)  {  var ratingBar = FindViewById<RatingBar>(Resource.Id.myRatingBar);  Toast.MakeText(this, ratingBar.Rating.ToString(CultureInfo.InvariantCulture), ToastLength.Long).Show();  }  }  } |

Gist file link: <https://gist.github.com/vkoppaka/6759fa7a4d2574d2fc94>



Venkata