

Mobile Application Development

User Interface

Common Input Controls

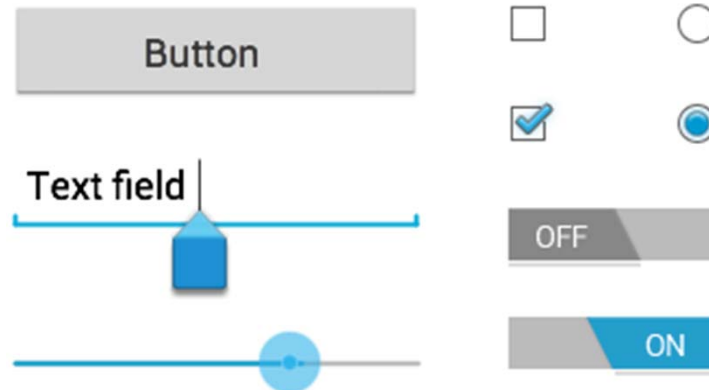
Common Input Controls

- Android provides a wide variety of controls you can use in your UI, such as [buttons](#), [text fields](#), [seek bars](#), [checkboxes](#), [toggle buttons](#), and many more.
- [Adding an input control](#) to your UI is as simple as adding an XML element to your XML layout.
- Each input control supports a specific set of [input events](#) so you can handle events such as when the user enters text or touches a button.

Common Input Controls

- Some common input controls:

- Buttons
- Text Fields
- Checkboxes
- Radio Buttons
- Toggle Buttons
- Seek Bars



BUTTONS

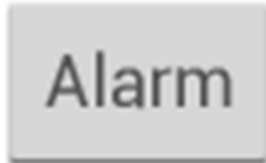
Buttons

- A **Button** consists of text and/or an image that clearly communicates what action will occur when the user touches it. A button can have an image, text, or both.



- To define the click event handler for a button, add the **android:onClick** attribute to the `<Button>` element in your XML layout.
- You can also **declare the click event handler programmatically** rather than in an XML layout.

Create Text Button (XML)



```
<Button  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/button_text"  
    ... />
```

Create Icon Button (XML)



```
<ImageButton  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:src="@drawable/button_icon"  
    ... />
```

Create Text+Icon Button (XML)



```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_text"
    android:drawableLeft="@drawable/button_icon"
    ... />
```


Button Click Events

- When the user clicks a button, the Button object receives an **on-click event**.
- To **define the click event handler** for a button, add the android:onClick attribute to the <Button> element in your XML layout.

android:onClick="sendMessage"

- The value for this attribute **must be the name of the method you want to call** in response to a click event.
- The Activity hosting the layout must then **implement the corresponding method**.

Define Click Event For Button (XML)

```
<Button  
    android:id="@+id/button_send"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/button_send"  
    android:onClick="sendMessage" />
```

Responding to Click Events (Code)

```
public void sendMessage(View view) {  
    // Do something in response to button click  
}
```

The method you declare in the android:onClick attribute must have a signature exactly as shown above. Specifically, the method must:

- Be **public**
- Return **void**
- Define a **View** as its only parameter (this will be the View that was clicked)

onClickListener (Code)

```
Button button = (Button) findViewById(R.id.button_send);

button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        // Do something in response to button click
    }
});
```

To declare the event handler programmatically, create a `View.OnClickListener` object and assign it to the button by calling `setOnClickListener(View.OnClickListener)`.

Button References

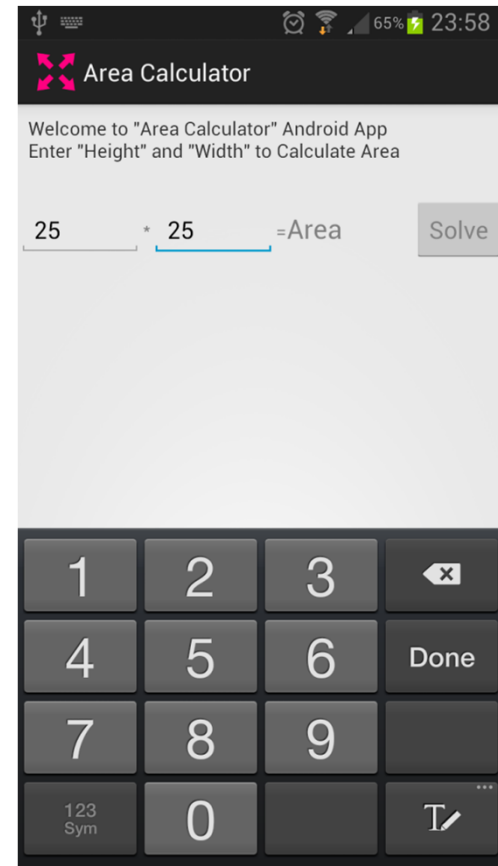
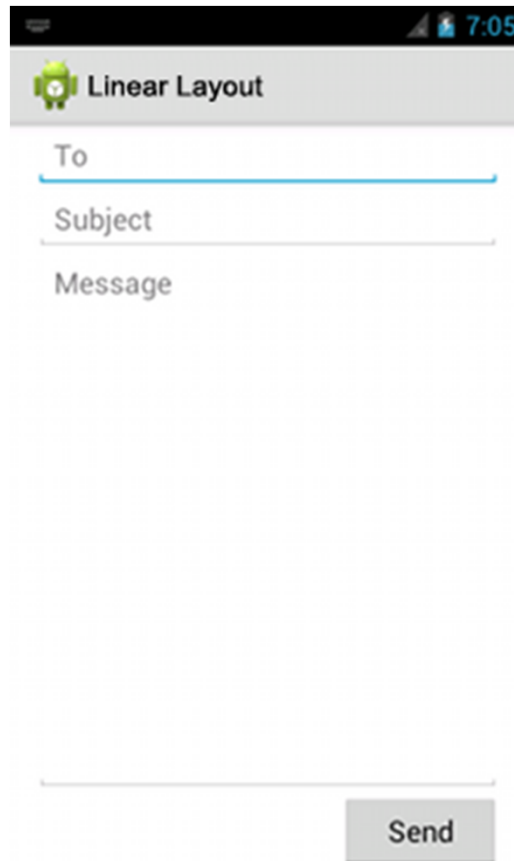
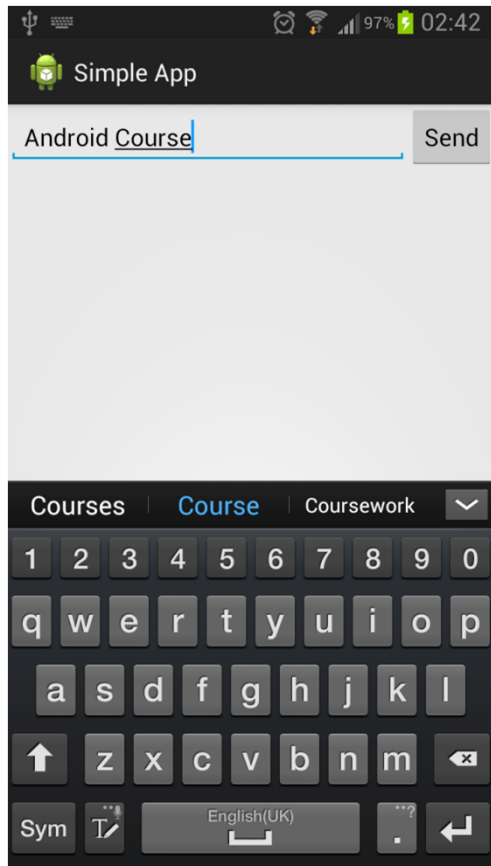
- <http://developer.android.com/guide/topics/ui/controls/button.html>
- <http://developer.android.com/reference/android/widget/Button.html>
- <http://developer.android.com/reference/android/widget/ImageButton.html>

TEXT FIELDS

Text Fields

- Text fields allow the user to **type text** into your app.
- They can be either **single line or multi-line**.
- Touching a text field places the cursor and **automatically displays the keyboard**.
- In addition to typing, text fields allow for a variety of other activities, such as **text selection (cut, copy, paste)** and data lookup via **auto-completion**.
- You can add a text field to you layout with the **EditText** object. You should usually do so in your XML layout with a `<EditText>` element.

Text Fields



Create EditText (XML)

```
<EditText  
    android:id="@+id/someID"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:hint="@string/some_hint" />
```

Specifying Keyboard (XML)

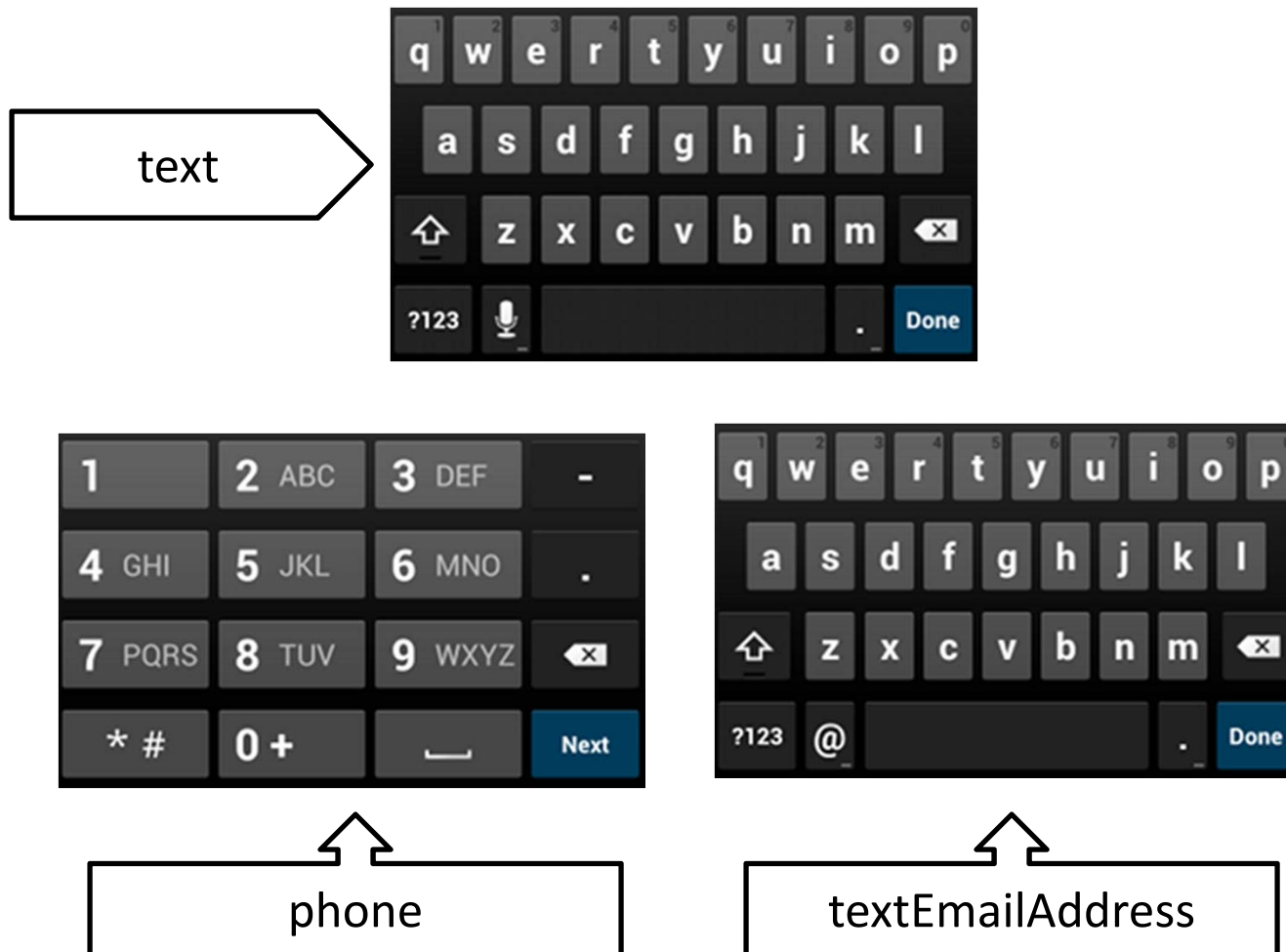
- You can specify the type of keyboard you want for your EditText object with the `android:inputType` attribute.
- For example, if you want the user to input an email address, you should use the `textEmailAddress` input type:

```
<EditText  
    android:id="@+id/someID"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:hint="@string/some_hint"  
    android:inputType="textEmailAddress" />
```

Specifying Keyboard (XML)

- There are several different input types available for different situations.
 - "text" Normal text keyboard.
 - "textEmailAddress" Normal text keyboard with the @ character.
 - "textUri" Normal text keyboard with the / character.
 - "number" Basic number keypad.
 - "phone" Phone-style keypad.

Specifying Keyboard (XML)



Keyboard Behavior (XML)

- The `android:inputType` also allows you to specify certain keyboard behaviors, such as `whether to capitalize all new words` or use features like auto-complete and spelling suggestions.

Keyboard Behavior (XML)

- The android:inputType attribute allows **bitwise combinations** so you can specify both a keyboard layout and one or more behaviors at once.
 - **"textMultiLine"** Normal text keyboard that allow users to input long strings of text that include line breaks (carriage returns).
 - **"textCapSentences"** Normal text keyboard that capitalizes the first letter for each new sentence.
 - **"textCapWords"** Normal text keyboard that capitalizes every word. Good for titles or person names.
 - **"textAutoCorrect"** Normal text keyboard that corrects commonly misspelled words.
 - **"textPassword"** Normal text keyboard, but the characters entered turn into dots.

Keyboard Behavior (XML)

```
<EditText  
    android:id="@+id/txt_special"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:hint="@string/txt_special_hint"  
    android:inputType="textMultiLine|textCapWords" />
```

Specifying Keyboard Actions

- In addition to changing the keyboard's input type, Android allows you to **specify an action** to be made when users have completed their input.
- The action specifies the **button that appears in place of the carriage return key** and the action to be made, such as "Search" or "Send."
- You can specify the action by setting the **android:imeOptions** attribute.
- IME stand for Input Method Editor.
- imeOptions: http://developer.android.com/reference/android/widget/TextView.html#attr_android:imeOptions

Specifying Keyboard Actions



Specifying Keyboard Actions (XML)

```
<EditText
    android:id="@+id/txt_special"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:hint="@string/txt_special_hint"
    android:inputType="text"
    android:imeOptions="actionSend" />
```

Specifying Keyboard Actions

- If you do not explicitly specify an input action then the system attempts to determine if there are any subsequent android:focusable fields. If any focusable fields are found following this one, the system applies the (`@code actionNext`) action to the current EditText so the user can select Next to move to the next field.
- If there's no subsequent focusable field, the system applies the "`actionDone`" action.

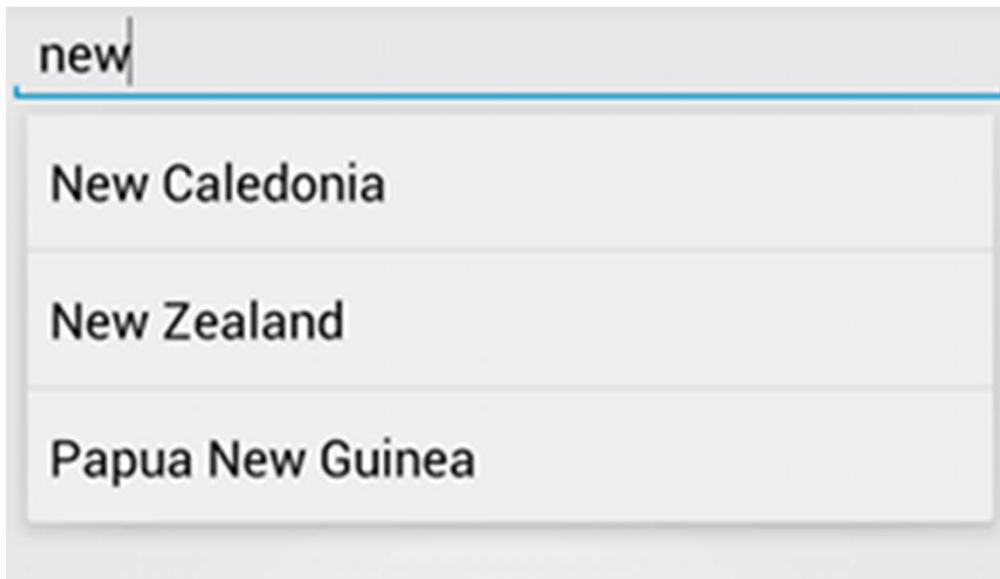
Responding to Keyboard Actions

```
EditText editText = (EditText) findViewById(R.id.txt_special);

editText.setOnEditorActionListener(new OnEditorActionListener() {
    @Override
    public boolean onEditorAction(TextView v, int actionId, KeyEvent event)
    {
        boolean handled = false;
        if (actionId == EditorInfo.IME_ACTION_SEND) {
            sendMessage();
            handled = true;
        }
        return handled;
    }
});
```

Providing Auto Complete

- If you want to [provide suggestions](#) to users as they type, you can use a subclass of EditText called [AutoCompleteTextView](#).



Providing Auto Complete

- To implement auto-complete, you must specify an **Adapter** that provides the text suggestions.
 - An **Adapter** object acts as a bridge between a View and the underlying data for that view.
 - The Adapter **provides access to the data items**.
 - The Adapter is also responsible for **making a View for each item** in the data set.
- There are several kinds of adapters available, depending on where the data is coming from, such as from a **database** or **an array**.

Providing Auto Complete

Steps:

1. Add the `AutoCompleteTextView` to your layout.
2. Define the array that contains all text suggestions. For example, an array of country names that's defined in an XML resource file (`res/values/strings.xml`):
3. In your Activity, specify the adapter that supplies the suggestions.

1. Add autoCompleteTextView

```
<?xml version="1.0" encoding="utf-8"?>
<AutoCompleteTextView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/autocomplete_country"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />
```


2. Define the Array

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="countries_array">
        <item>Afghanistan</item>
        <item>Albania</item>
        <item>Algeria</item>
        <item>American Samoa</item>
        <item>Andorra</item>
        <item>Angola</item>
        <item>Anguilla</item>
        <item>Antarctica</item>
        ...
    </string-array>
</resources>
```

3. Specify the Adapter

```
// Get a reference to the autoCompleteTextView in the layout
AutoCompleteTextView textView = (AutoCompleteTextView)
    findViewById(R.id.autocomplete_country);

// Get the string array
String[] countries =
    getResources().getStringArray(R.array.countries_array);

// Create the adapter and set it to the autoCompleteTextView
ArrayAdapter<String> adapter =
    new ArrayAdapter<String>(this,
        android.R.layout.simple_list_item_1, countries);

textView.setAdapter(adapter);
```

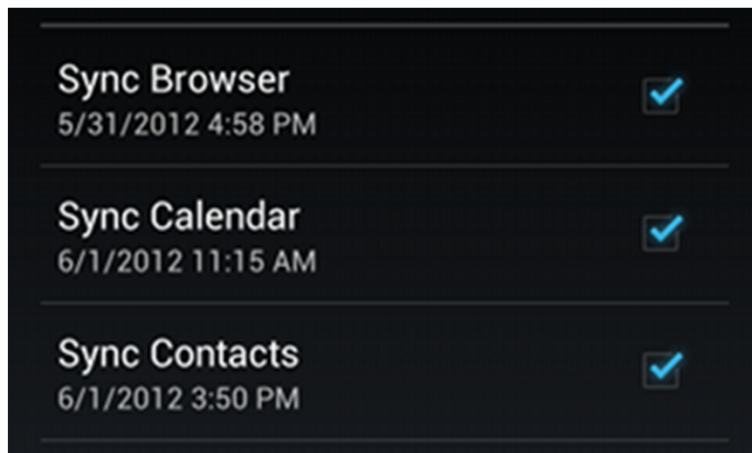
Text Field References

- <http://developer.android.com/guide/topics/ui/controls/text.html>
- <http://developer.android.com/reference/android/widget/EditText.html>
- <http://developer.android.com/design/building-blocks/text-fields.html>
- <http://developer.android.com/guide/topics/text/creating-input-method.html>
- <http://developer.android.com/reference/android/widget/AutoCompleteTextView.html>
- <http://developer.android.com/reference/android/widget/ArrayAdapter.html>
- <http://developer.android.com/reference/android/R.layout.html>
- <http://docs.oracle.com/javase/tutorial/java/generics/types.html>

CHECKBOXES

Checkboxes

- Checkboxes allow the user to [select one or more options](#) from a set.
- Typically, you should present each checkbox option in a [vertical list](#).



- Because a set of checkbox options allows the user to select multiple items, each checkbox is managed separately and you [must register a click listener for each one](#).

Create Checkbox (XML)

```
<CheckBox android:id="@+id/checkbox_meat"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/meat"  
    android:onClick="onCheckboxClicked" />  
<CheckBox android:id="@+id/checkbox_cheese"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/cheese"  
    android:onClick="onCheckboxClicked" />
```

Responding to Click Events

- When the user selects a checkbox, the CheckBox object receives an **on-click event**.
- The value for this attribute must be the **name of the method** you want to call in response to a click event.
- The Activity hosting the layout must then **implement the corresponding method**.
- The method you declare in the android:onClick attribute must:
 - Be **public**
 - Return **void**
 - Define a **View** as its only parameter (this will be the View that was clicked)

Responding to Click Events (Code)

```
public void onCheckboxClicked(View view) {  
    boolean checked = ((CheckBox) view).isChecked();  
    switch(view.getId()) {  
        case R.id.checkbox_meat:  
            if (checked)  
                ...  
            else  
                ...  
            break;  
        case R.id.checkbox_cheese:  
            if (checked)  
                ...  
            else  
                ...  
            break;  
    }  
}
```


Accessing CheckBox (Code)

```
CheckBox cb1 = (CheckBox) findViewById(R.id. checkbox_meat);
```

```
if (cb1.isChecked()) {  
    ...  
} else {  
    ...  
}
```

```
CheckBox cb2 = (CheckBox) findViewById(R.id. checkbox_cheese);
```

```
if (cb2.isChecked()) {  
    ...  
} else {  
    ...  
}
```

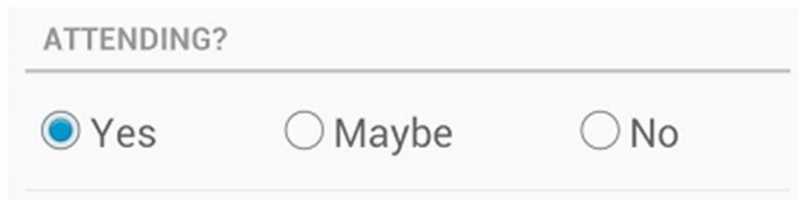
Checkboxes References

- <http://developer.android.com/guide/topics/ui/controls/checkbox.html>
- <http://developer.android.com/reference/android/widget/CheckBox.html>

RADIO BUTTONS

Radio Buttons

- Radio buttons allow the user to [select one option from a set](#). You should use radio buttons for optional sets that are mutually exclusive if you think that the user needs to see all available options side-by-side.



ATTENDING?

☒ Yes ☐ Maybe ☐ No

- To create each radio button option, create a [RadioButton](#) in your layout.
- However, because [radio buttons are mutually exclusive](#), you must group them together inside a [RadioGroup](#).
- By grouping them together, the system ensures that only one radio button can be selected at a time.

Create Radio Buttons (XML)

```
<RadioGroup
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">
    <RadioButton android:id="@+id/radio_pirates"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/pirates"
        android:onClick="onRadioButtonClicked" />
    <RadioButton android:id="@+id/radio_ninjas"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/ninjas"
        android:onClick="onRadioButtonClicked" />
</RadioGroup>
```

Responding to Click Events

- When the user selects one of the radio buttons, the corresponding `RadioButton` object receives an `on-click event`.
- To define the click event handler for a button, add the `android:onClick` attribute to the `<RadioButton>` element in your XML layout.
- The value for this attribute must be the name of the method you want to call in response to a click event. The Activity hosting the layout must then `implement the corresponding method`.
- The method you declare in the `android:onClick` attribute must:
 - Be `public`
 - Return `void`
 - Define a `View` as its only parameter (this will be the View that was clicked)

Responding to Click Events (Code)

```
public void onRadioButtonClicked(View view) {  
    boolean checked = ((RadioButton) view).isChecked();  
  
    switch(view.getId()) {  
        case R.id.radio_pirates:  
            if (checked)  
                // Pirates are the best  
            break;  
        case R.id.radio_ninjas:  
            if (checked)  
                // Ninjas rule  
            break;  
    }  
}
```

Accessing RadioButton (Code)

```
RadioButton rb1 = (RadioButton) findViewById(R.id. radio_pirates);
```

```
if (rb1.isChecked()) {  
    ...  
}
```

```
RadioButton rb1 = (RadioButton) findViewById(R.id. radio_ninjas);
```

```
if (rb2.isChecked()) {  
    ...  
}
```


Radio Button References

- <http://developer.android.com/guide/topics/ui/controls/radiobutton.html>
- <http://developer.android.com/reference/android/widget/RadioButton.html>
- <http://developer.android.com/reference/android/widget/RadioGroup.html>

TOGGLE BUTTONS

Toggle Buttons

- A toggle button allows the user to change a setting between two states.
- You can add a basic toggle button to your layout with the ToggleButton object.



- Android 4.0 (API level 14) introduces another kind of toggle button called a switch that provides a slider control, which you can add with a Switch object.



- The ToggleButton and Switch controls are subclasses of CompoundButton and function in the same manner, so you can implement their behavior the same way.

Create ToggleButtons (XML)

```
<ToggleButton  
    android:id="@+id/togglebutton"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:textOn="Vibrate on"  
    android:textOff="Vibrate off"  
    android:onClick="onToggleClicked" />
```

Toggle Buttons

- When the user selects a ToggleButton and Switch, the object receives an [on-click event](#).
- To [define the click event handler](#), add the android:onClick attribute to the <ToggleButton> or <Switch> element in your XML layout.
- The value for this attribute must be the name of the method you want to call in response to a click event. The Activity hosting the layout must then [implement the corresponding method](#).
- The method you declare in the android:onClick attribute must:
 - Be [public](#)
 - Return [void](#)
 - Define a [View](#) as its only parameter (this will be the View that was clicked)

Responding to Click Events (Code)

```
public void onToggleClicked(View view) {  
    // Is the toggle on?  
    boolean on = ((ToggleButton) view).isChecked();  
  
    if (on) {  
        // Enable vibrate  
    } else {  
        // Disable vibrate  
    }  
}
```

OnCheckedChangeListener (Code)

```
ToggleButton toggle = (ToggleButton) findViewById(R.id.togglebutton);

toggle.setOnCheckedChangeListener(new
    CompoundButton.OnCheckedChangeListener() {
    public void onCheckedChanged(CompoundButton buttonView,
        boolean isChecked)
    {
        if (isChecked) {
            // do something
        } else {
            // do something
        }
    }
});
```

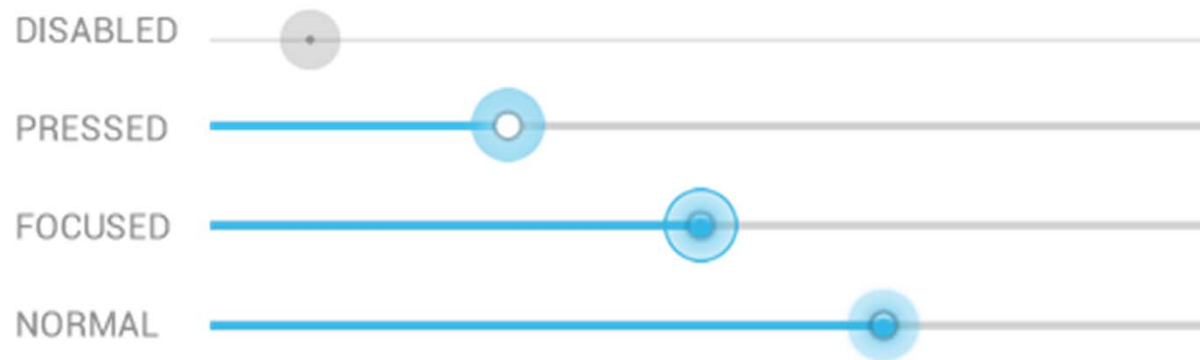
Toggle Button References

- <http://developer.android.com/guide/topics/ui/controls/togglebutton.html>
- <http://developer.android.com/reference/android/widget/ToggleButton.html>
- <http://developer.android.com/reference/android/widget/Switch.html>

SEEK BARS

Seek Bars

- Using [SeekBar](#) control, users can set the current value (progress level) by sliding the thumb left or right.



- Clients of the SeekBar can attach a [SeekBar.OnSeekBarChangeListener](#) to be notified of the user's actions.

Create SeekBar (XML)

```
<SeekBar  
    android:id="@+id/testSeekBar"  
    android:  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:progress="30"  
    android:max="100" />
```

Access SeekBar (Code)

```
SeekBar sb=(SeekBar) findViewById(R.id.testSeekBar);
```

```
int v, max;
```

```
v = sb.getProgress();
```

```
max = sb.getMax();
```

```
TextView tv=(TextView) findViewById(R.id.value);
```

```
tv.setText(String.valueOf(v));
```

SeekBar Event Listener

```
sb.setOnSeekBarChangeListener(new OnSeekBarChangeListener() {  
    int progress = 0;  
    @Override  
    public void onProgressChanged(SeekBar seekBar, int progresValue, boolean fromUser)  
    {  
        progress = progresValue;  
    }  
    @Override  
    public void onStartTrackingTouch(SeekBar seekBar) {  
        // Do something here, if you want to do anything at the start of  
        // touching the seekbar  
    }  
    @Override  
    public void onStopTrackingTouch(SeekBar seekBar) {  
        // Display the value in textview (tv must be final)  
        tv.setText(progress + "/" + seekBar.getMax());  
    }  
});
```

SeekBar References

- <http://developer.android.com/design/building-blocks/seek-bars.html>
- <http://developer.android.com/reference/android/widget/SeekBar.html>
- <http://developer.android.com/reference/android/widget/SeekBar.OnSeekBarChangeListener.html>

Q & A