Form Stuff

This tutorial introduces a variety of widgets that are useful when creating forms, such as image buttons, text fields, checkboxes and radio buttons.

1. Start a new project named *HelloFormStuff*.
2. Your `res/layout/main.xml` file should already have a basic `LinearLayout`:

   ```
   <?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" >
   </LinearLayout>
   ```

   For each widget you want to add, just put the respective View inside this `LinearLayout`.

Each section below also assumes that your `HelloFormStuff` Activity has the following default implementation of the `onCreate()` method:

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
}
```

Now select which kind of form widget you'd like to create:

- Custom Button
- Edit Text
- Checkbox
- Radio Buttons
- Toggle Button
- Rating Bar

Custom Button

In this section, you will create a button with a custom image instead of text, using the `Button` widget and an XML file that defines three different images to use for the different button states. When the button is pressed, a short message will be displayed.

1. Copy the images on the right into the `res/drawable/` directory of your project. These will be used for the different button states.
2. Create a new file in the `res/drawable/` directory named `android_button.xml`. Insert the following XML:

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable="@drawable/android_pressed"
          android:state_pressed="true" />
    <item android:drawable="@drawable/android_focused"
          android:state_focused="true" />
    <item android:drawable="@drawable/android_normal" />
</selector>
```

This defines a single drawable resource, which will change its image based on the current state of the button. The first `<item>` defines `android_pressed.png` as the image when the button is pressed (it's been activated); the second `<item>` defines `android_focused.png` as the image when the button is focused (when the button is highlighted using the trackball or directional pad); and the third `<item>` defines `android_normal.png` as the image for the normal state (when neither pressed nor focused). This XML file now represents a single drawable resource and when referenced by a `Button` for its background, the image displayed will change based on these three states.

> **Note:** The order of the `<item>` elements is important. When this drawable is referenced, the `<item>`s are traversed in-order to determine which one is appropriate for the current button state. Because the "normal" image is last, it is only applied when the conditions `android:state_pressed` and `android:state_focused` have both evaluated false.

3. Open the `res/layout/main.xml` file and add the `Button` element:

```
    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="10dp"
        android:background="@drawable/android_button" />
```

The `android:background` attribute specifies the drawable resource to use for the button background (which, when saved at `res/drawable/android.xml`, is referenced as `@drawable/android`). This replaces the normal background image used for buttons throughout the system. In order for the drawable to change its image based on the button state, the image must be applied to the background.

4. To make the button do something when pressed, add the following code at the end of the `onCreate()` method:

```
final Button button = (Button) findViewById(R.id.button);
button.setOnClickListener(new OnClickListener() {
    public void onClick(View v) {
        // Perform action on clicks
        Toast.makeText(HelloFormStuff.this, "Beep Bop", Toast.LENGTH_SHORT).show
();
    }
});
```

This captures the `Button` from the layout, then adds an `View.OnClickListener`. The `View.OnClickListener` must implement the `onClick(View)` callback method, which defines the action to be made when the button is clicked. In this example, a `Toast` message will be displayed.

5. Now run the application.

Edit Text

In this section, you will create a text field for user input, using the `EditText` widget. Once text has been entered into the field, the "Enter" key will display the text in a toast message.

1. Open the `res/layout/main.xml` file and add the `EditText` element (inside the `LinearLayout`):

```
    <EditText
        android:id="@+id/edittext"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"/>
```

2. To do something with the text that the user types, add the following code to the end of the `onCreate()` method:

```
final EditText edittext = (EditText) findViewById(R.id.edittext);
edittext.setOnKeyListener(new OnKeyListener() {
    public boolean onKey(View v, int keyCode, KeyEvent event) {
        // If the event is a key-down event on the "enter" button
        if ((event.getAction() == KeyEvent.ACTION_DOWN) &&
            (keyCode == KeyEvent.KEYCODE_ENTER)) {
          // Perform action on key press
          Toast.makeText(HelloFormStuff.this, edittext.getText(),
Toast.LENGTH_SHORT).show();
          return true;
        }
        return false;
    }
});
```

   This captures the `EditText` element from the layout and adds an `View.OnKeyListener`. The `View.OnKeyListener` must implement the `onKey(View, int, KeyEvent)` method, which defines the action to be made when a key is pressed while the widget has focus. In this case, the method is defined to listen for the Enter key (when pressed down), then pop up a `Toast` message with the text that has been entered. The `onKey(View, int, KeyEvent)` method should always return `true` if the event has been handled, so that the event doesn't bubble-up (which would result in a carriage return in the text field).

3. Run the application.

Checkbox

In this section, you will create a checkbox for selecting items, using the `CheckBox` widget. When the checkbox is pressed, a toast message will indicate the current state of the checkbox.

1. Open the `res/layout/main.xml` file and add the `CheckBox` element (inside the `LinearLayout`):

```
    <CheckBox android:id="@+id/checkbox"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="check it out" />
```

2. To do something when the state is changed, add the following code to the end of the `onCreate()` method:

```
final CheckBox checkbox = (CheckBox) findViewById(R.id.checkbox);
checkbox.setOnClickListener(new OnClickListener() {
    public void onClick(View v) {
        // Perform action on clicks, depending on whether it's now checked
        if (((CheckBox) v).isChecked()) {
            Toast.makeText(HelloFormStuff.this, "Selected",
Toast.LENGTH_SHORT).show();
        } else {
            Toast.makeText(HelloFormStuff.this, "Not selected",
Toast.LENGTH_SHORT).show();
        }
    }
});
```

This captures the `CheckBox` element from the layout, then adds an `View.OnClickListener`. The `View.OnClickListener` must implement the `onClick(View)` callback method, which defines the action to be made when the checkbox is clicked. When clicked, `isChecked()` is called to check the new state of the check box. If it has been checked, then a `Toast` displays the message "Selected", otherwise it displays "Not selected". Note that the `View` object that is passed in the `onClick(View)` callback must be cast to a `CheckBox` because the `isChecked()` method is not defined by the parent `View` class. The `CheckBox` handles its own state changes, so you only need to query the current state.

3. Run it.

> **Tip:** If you need to change the state yourself (such as when loading a saved `CheckBoxPreference`), use the `setChecked(boolean)` or `toggle()` method.

Radio Buttons

In this section, you will create two mutually-exclusive radio buttons (enabling one disables the other), using the `RadioGroup` and `RadioButton` widgets. When either radio button is pressed, a toast message will be displayed.

1. Open the `res/layout/main.xml` file and add two `RadioButton`s, nested in a `RadioGroup` (inside the `LinearLayout`):

```
    <RadioGroup
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:orientation="vertical">
      <RadioButton android:id="@+id/radio_red"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="Red" />
      <RadioButton android:id="@+id/radio_blue"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="Blue" />
    </RadioGroup>
```

It's important that the `RadioButton`s are grouped together by the `RadioGroup` element so that no more than one can be selected at a time. This logic is automatically handled by the Android system. When one `RadioButton` within a group is selected, all others are automatically deselected.

2. To do something when each `RadioButton` is selected, you need an `View.OnClickListener`. In this case, you want the listener to be re-usable, so add the following code to create a new member in the `HelloFormStuff` Activity:

```
private OnClickListener radio_listener = new OnClickListener() {
    public void onClick(View v) {
        // Perform action on clicks
        RadioButton rb = (RadioButton) v;
        Toast.makeText(HelloFormStuff.this, rb.getText(),
Toast.LENGTH_SHORT).show();
    }
};
```

   First, the `View` that is passed to the `onClick(View)` method is cast into a RadioButton. Then a `Toast` message displays the selected radio button's text.

3. Now, at the bottom of the `onCreate()` method, add the following:

```
  final RadioButton radio_red = (RadioButton) findViewById(R.id.radio_red);
  final RadioButton radio_blue = (RadioButton) findViewById(R.id.radio_blue);
  radio_red.setOnClickListener(radio_listener);
  radio_blue.setOnClickListener(radio_listener);
```

   This captures each of the `RadioButton`s from the layout and adds the newly-created `View.OnClickListener` to each.

4. Run the application.

> **Tip:** If you need to change the state yourself (such as when loading a saved `CheckBoxPreference`), use the `setChecked(boolean)` or `toggle()` method.

Toggle Button

In this section, you'll create a button used specifically for toggling between two states, using the `ToggleButton` widget. This widget is an excellent alternative to radio buttons if you have two simple states that are mutually exclusive ("on" and "off", for example).

1. Open the `res/layout/main.xml` file and add the `ToggleButton` element (inside the `LinearLayout`):

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

   The attributes `android:textOn` and `android:textOff` specify the text for the button when the button has been toggled on or off. The default values are "ON" and "OFF".

2. To do something when the state is changed, add the following code to the end of the `onCreate()` method:

```
final ToggleButton togglebutton = (ToggleButton) findViewById(R.id.togglebutton);
togglebutton.setOnClickListener(new OnClickListener() {
    public void onClick(View v) {
        // Perform action on clicks
        if (togglebutton.isChecked()) {
```

```
            Toast.makeText(HelloFormStuff.this, "Checked",
Toast.LENGTH_SHORT).show();
        } else {
            Toast.makeText(HelloFormStuff.this, "Not checked",
Toast.LENGTH_SHORT).show();
        }
    }
});
```

This captures the `ToggleButton` element from the layout, then adds an `View.OnClickListener`. The `View.OnClickListener` must implement the `onClick(View)` callback method, which defines the action to perform when the button is clicked. In this example, the callback method checks the new state of the button, then shows a `Toast` message that indicates the current state.

Notice that the `ToggleButton` handles its own state change between checked and unchecked, so you just ask which it is.

3. Run the application.

> **Tip:** If you need to change the state yourself (such as when loading a saved `CheckBoxPreference`), use the `setChecked(boolean)` or `toggle()` method.

Rating Bar

In this section, you'll create a widget that allows the user to provide a rating, with the `RatingBar` widget.

1. Open the `res/layout/main.xml` file and add the `RatingBar` element (inside the `LinearLayout`):

```
    <RatingBar android:id="@+id/ratingbar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:numStars="5"
        android:stepSize="1.0"/>
```

The `android:numStars` attribute defines how many stars to display for the rating bar. The `android:stepSize` attribute defines the granularity for each star (for example, a value of `0.5` would allow half-star ratings).

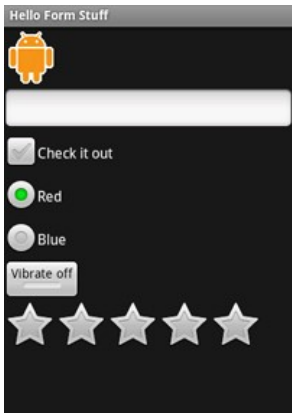
2. To do something when a new rating has been set, add the following code to the end of the `onCreate()` method:

```
final RatingBar ratingbar = (RatingBar) findViewById(R.id.ratingbar);
ratingbar.setOnRatingBarChangeListener(new OnRatingBarChangeListener() {
    public void onRatingChanged(RatingBar ratingBar, float rating, boolean
fromUser) {
        Toast.makeText(HelloFormStuff.this, "New Rating: " + rating,
Toast.LENGTH_SHORT).show();
    }
});
```

This captures the `RatingBar` widget from the layout with `findViewById(int)` and then sets an `RatingBar.OnRatingBarChangeListener`. The `onRatingChanged()` callback method then defines the action to perform when the user sets a rating. In this case, a simple `Toast` message displays the new rating.

3. Run the application.

If you've added all the form widgets above, your application should look like this:



References

- `ImageButton`
- `EditText`
- `CheckBox`
- `RadioButton`
- `ToggleButton`
- `RatingBar`

← Back to Hello, Views

↑ Go to top