+ Queens University Belfast

Android Studio

CSC3054 / CSC7054

XML Attributes

XML Attributes

A view in Android represents a widget e.g. a button or a layout manager. The Android SDK provides standards views (widgets) e.g. via the `TextView`, `EditText`, `Button` and `ImageView` classes. This tutorial will focus on the standard widgets and their properties and will consider two layouts `RelativeLayout` and `LinearLayout`.

Before You Begin

Open `Android Studio` and create a new project called `"Working with XML Properties"`. Refer to the `'Creating your first project'` tutorial to help you create a project. Once created your project should look like figure 1. Switch from `Design` view to Text `view`.

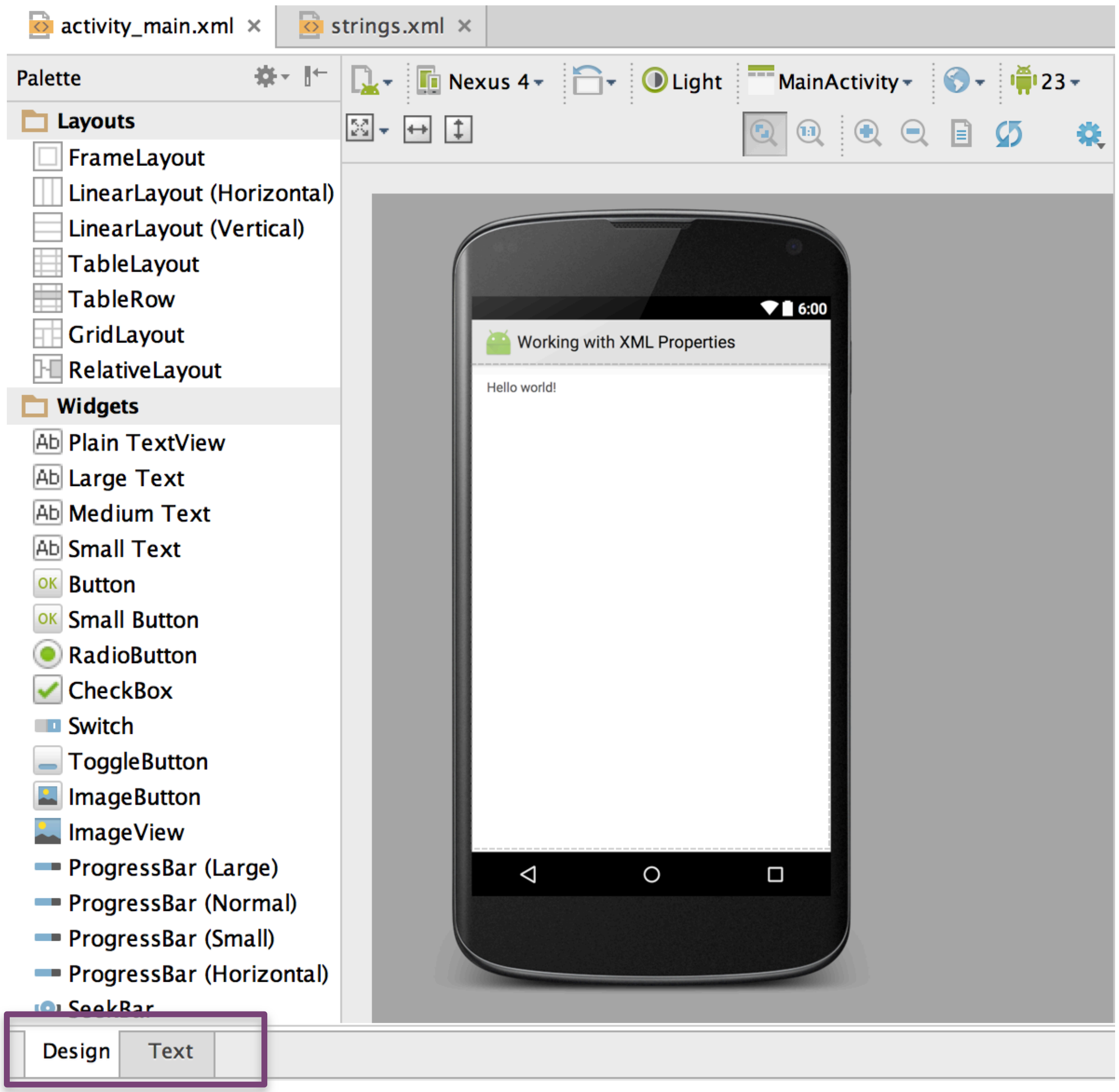
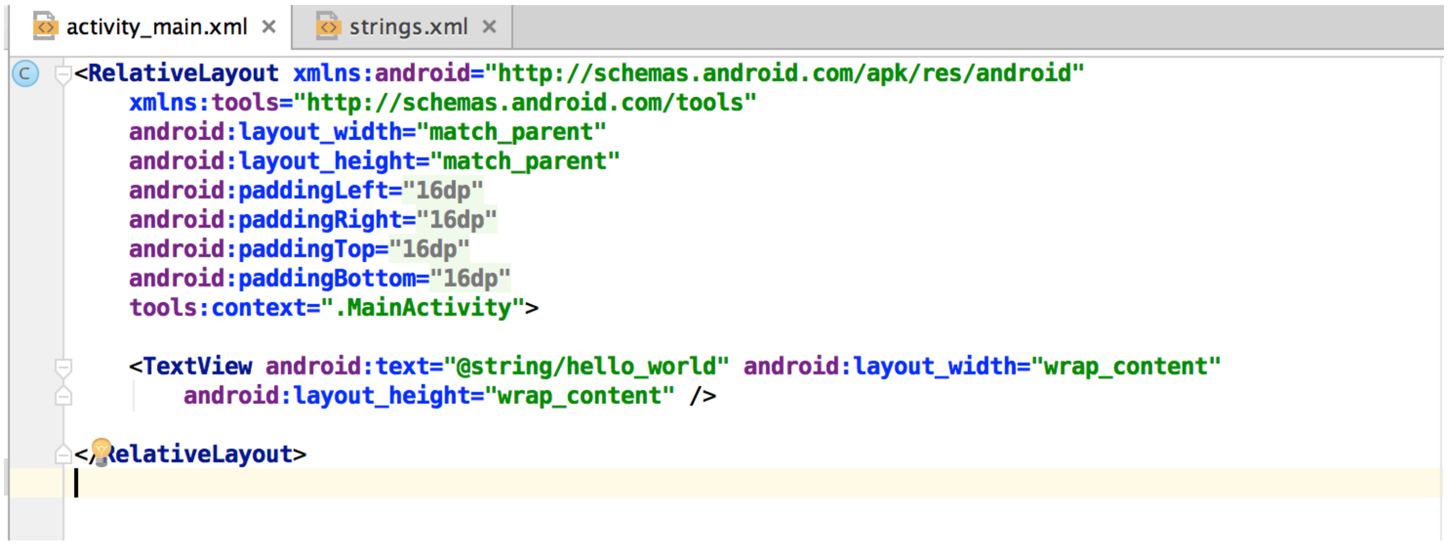


FIGURE 1 - OPEN PROJECT

This shall bring you to the XML view of the canvas as shown in figure 2.



```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    tools:context=".MainActivity">

    <TextView android:text="@string/hello_world" android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</RelativeLayout>
```

FIGURE 2 - XML (CODE VIEW) OF COMPONENTS

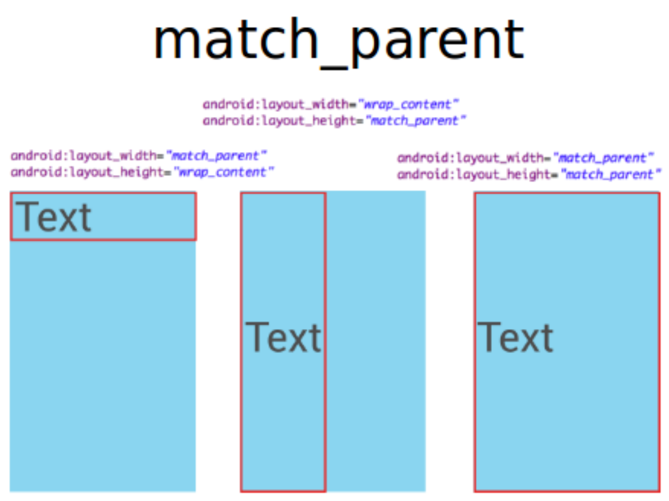
EXERCISE 1 – Working with the layout

Layout Manager

A layout manager is a subclass of `ViewGroup` and is responsible for the layout itself and its child Views. The most relevant layout managers are: `LinearLayout`, `FrameLayout`, `RelativeLayout` and `GridLayout`.

All layouts allow the developer to define attributes. The desired height and width can be define via the following attributes:

Attribute	Description
`android:layout_width`	Defines the width of the widget
`android:layout_height`	Defines the height of the widget
`android:layout_width="100dp"`	Widgets can use fixed sizes e.g. with the `dp` definition e.g. `100dp`. While `dp` is a fixed size it will scale with different device configurations.
`android: layout_width = "match_parent"`	Tells the application to maximize the widget in its parent. Figure 3
`android:layout_height ="wrap_content"`	Tells the layout to allocate the minimum amount so that the widget is rendered correctly. Figure 4

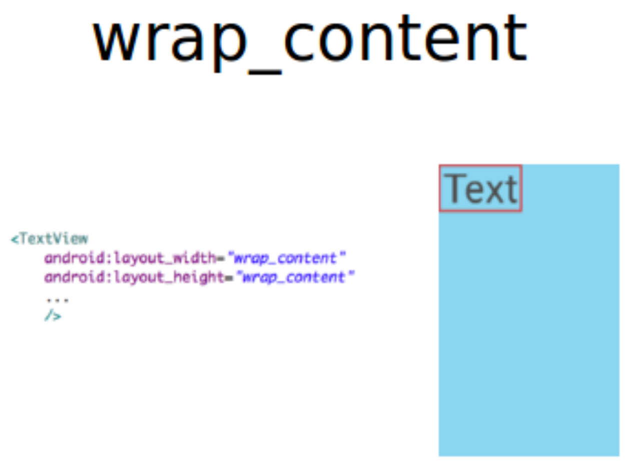


```
android:layout_width="wrap_content"
android:layout_height="match_parent"
```

```
android:layout_width="match_parent"
android:layout_height="wrap_content"
```

```
android:layout_width="match_parent"
android:layout_height="match_parent"
```

FIGURE 3- MATCH_PARENT



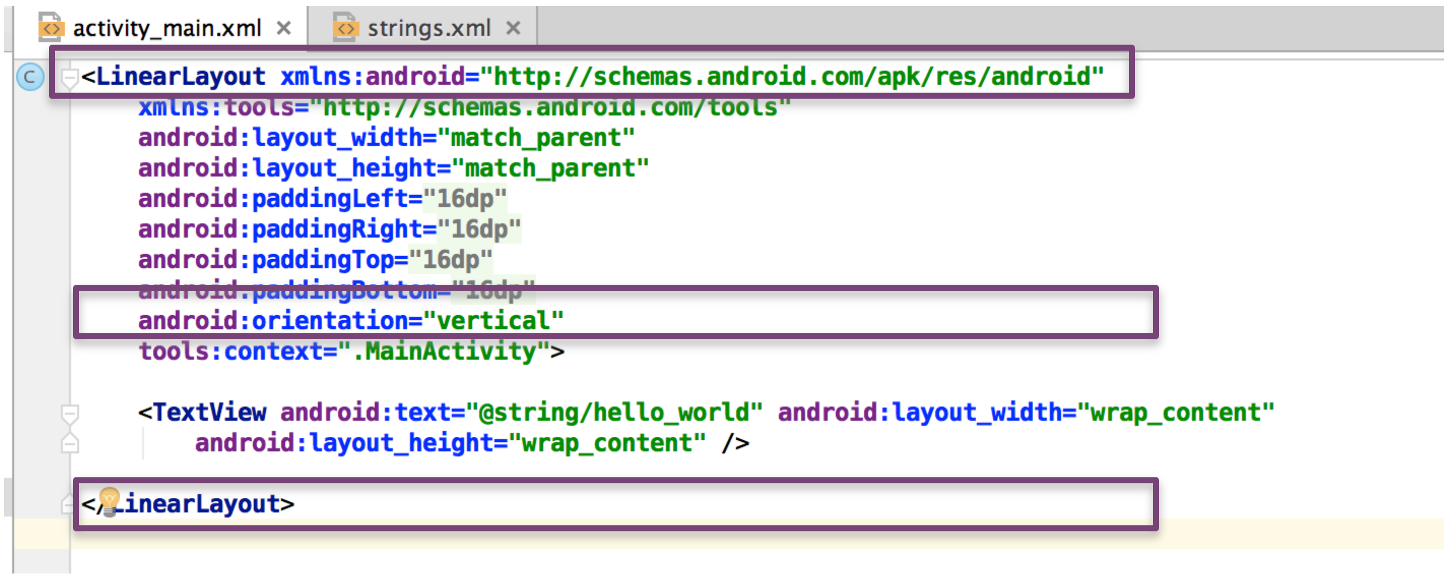
```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    ...
    />
```

FIGURE 4 WRAP_CONTENT

Task 1

You should notice that the layout of the canvas is set to `RelativeLayout` which allows you to place the widgets relative . This can be used for complex layouts, which are resource intensive. This tutorial is going to use a simpler layout called `LinearLayout`, which puts all its widgets into a single column or row depending on the `android:orientation` attribute. Possible values for this attribute are `horizontal` and `vertical`.

Change the layout manager from a `RelativeLayout` to a `LinearLayout` and set the orientation to `vertical` so all elements are put on a new row as per figure 5.



```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <TextView android:text="@string/hello_world" android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</LinearLayout>
```

FIGURE 5 - CHANGING THE LAYOUT

EXERCISE 2 – Working with the `TextView` Attribute

Task 1: The `typeface` Attribute

Attribute	Possible Values	Additional Notes
`android:typeface`	`sans`	
	`monospace`	
	`serif`	
	`normal`	This defaults to the `sans` typeface.

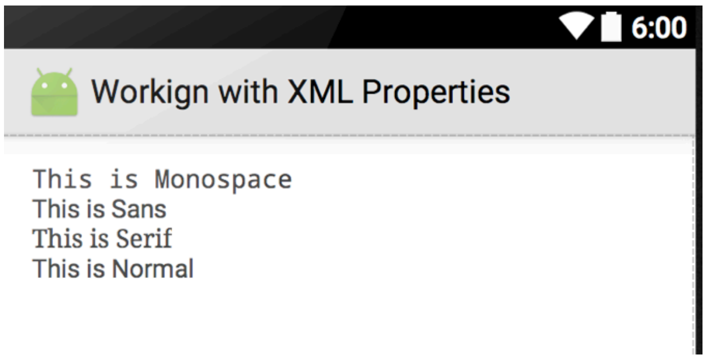
Create 4 new `TextViews` in the XML. Using the `android:typeface` attribute, specify a different value for each `TextView` the in the XML. When rendered, it should look like figure 6.

```
<TextView android:text="This is Monospace"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:typeface="monospace"/>

<TextView android:text="This is Sans"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:typeface="sans"/>

<TextView android:text="This is Serif"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:typeface="serif"/>

<TextView android:text="This is Normal"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:typeface="normal"/>
```



```
This is Monospace
This is Sans
This is Serif
This is Normal
```

FIGURE 6 - TYPEFACE ATTRIBUTE

Task 2 - `textStyle` Attribute

Attribute	Possible Values	Additional Notes
android:textStyle	normal	
	bold	
	italic	
	bold|italic	

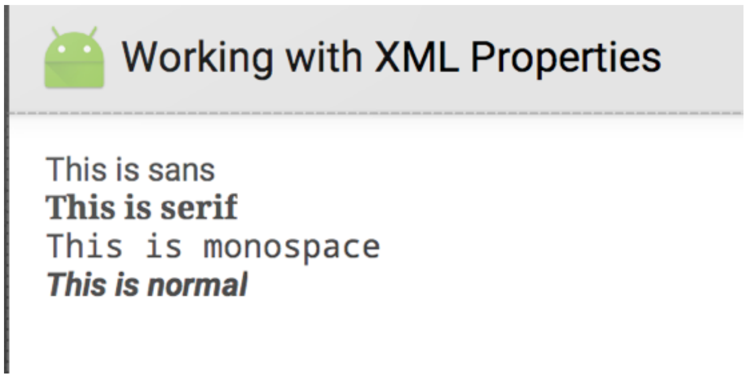
Add the `android:textStyle` attribute to each `TextView`. Specify a different value for each `TextView` the in the XML. When rendered, it should look like figure 7.

```
 <TextView
    android:text="This is sans"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:typeface="sans"
    android:textStyle="normal"/>

<TextView
    android:text="This is serif"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:typeface="serif"
    android:textStyle="bold"/>

<TextView
    android:text="This is monospace"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:typeface="monospace"
    android:textStyle="italic"/>

<TextView
    android:text="This is normal"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:typeface="normal"
    android:textStyle="bold|italic"/>
```


Working with XML Properties

This is sans
This is serif
This is monospace
This is normal

FIGURE 7 - TEXTSTYLE ATTRIBUTE

Task 3 - `textSize` Attribute

Attribute	Possible Values	Additional Notes
`android:textSize` Specifies the font size. Floating-point Number + Unit	`12sp`	It is generally a good practice to use the `sp` unit so the size can scale depending on user settings.
	`14sp`	
	`16sp`	
	`20sp`	

Too many type sizes and styles at once can wreck any layout. The basic sets of styles are based on a typographic scale of 12, 14, 16, 20, and 34. Refer to this typography styles guide for more details.

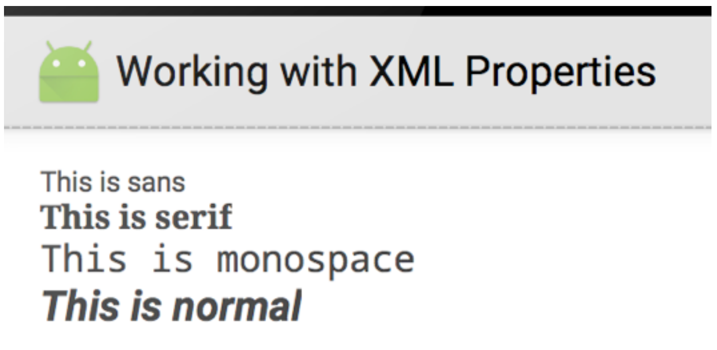
Add the `android:textSize` attribute to each `TextView`. Specify a different value for each `TextView` the in the XML. When rendered, it should look like figure 8.

```
<TextView
    android:text="This is sans"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:typeface="sans"
    android:textStyle="normal"
    android:textSize="12sp"/>

<TextView
    android:text="This is serif"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:typeface="serif"
    android:textStyle="bold"
    android:textSize="14sp"/>

<TextView
    android:text="This is monospace"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:typeface="monospace"
    android:textStyle="italic"
    android:textSize="16sp"/>

<TextView
    android:text="This is normal"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:typeface="normal"
    android:textStyle="bold|italic"
    android:textSize="18sp"/>
```



This is sans
This is serif
This is monospace
This is normal

FIGURE 8 - TEXTSIZE ATTRIBUTE

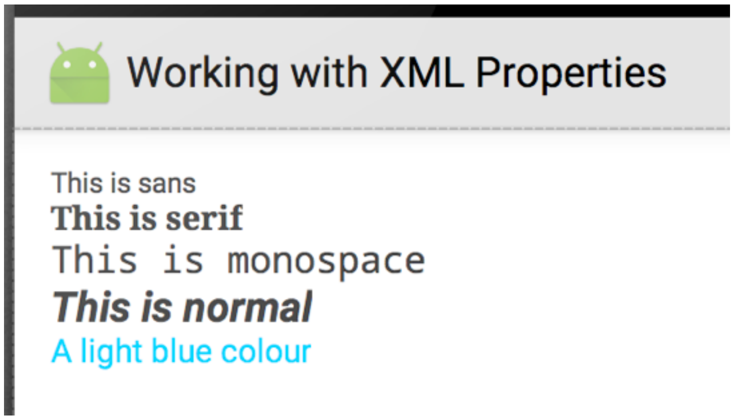
Task 4 – `textColor` Attribute

This attribute values are hexadecimal RGB values with an optional alpha channel, similar to what's found in CSS.

Attribute	Description
`android:textColor`	This attribute changes the color of the text
`android:textColorLink`	This attribute controls the highlighting for hyperlinks embedded within the `TextView`.

Add an additional TextView into the XML. When rendered it should look like figure 9.

```
<TextView
    android:text="A light blue colour"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textColor="#00ccff"
    android:textColorLink="#8DE67F" />
```


Working with XML Properties

This is sans
This is serif
This is monospace
This is normal
A light blue colour

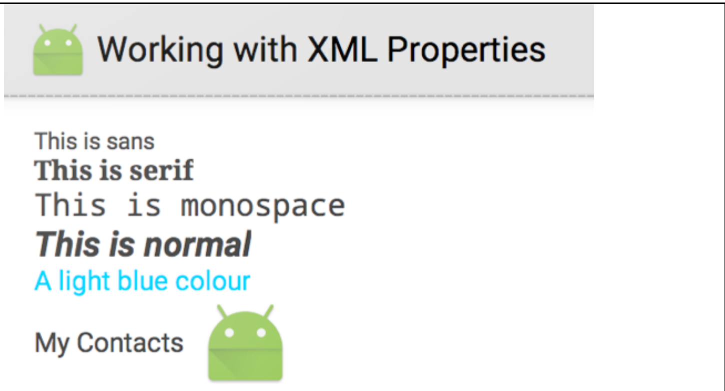
FIGURE 9 - TEXTCOLOR ATTRIBUTE

Task 5 – Displaying Images within a `TextView`

A `TextView` is actually surprisingly powerful and actually supports having images displayed as a part of it's content area. Any images stored in the "`mipmap`" folders can actually be embedded within a `TextView` at several key locations in relation to the text. The relevant attributes here are `drawableLeft`, `drawableRight`, `drawableTop` and `drawableBottom` along with `drawablePadding`.

Use the `android:drawableRight` and the `android:drawablePadding` attributes to insert the android icon to the right of the text in the `TextView`. When rendered it should look like figure 10.

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:gravity="center"
    android:text="My Contacts"
    android:drawableRight="@mipmap/ic_launcher"
    android:drawablePadding="8dp"
    />
```



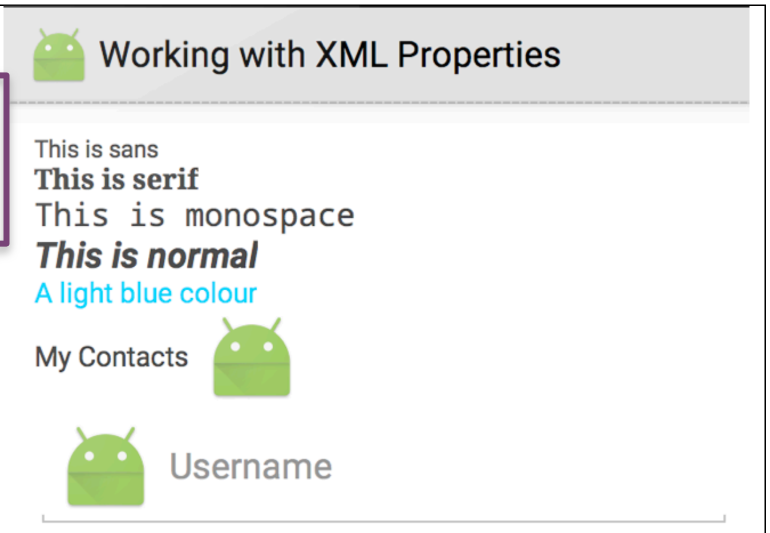
Working with XML Properties

This is sans
This is serif
This is monospace
This is normal
A light blue colour
My Contacts

FIGURE 10 – DISPLAYING IMAGES WITHIN A TEXTVIEW

In Android, many views inherit from `TextView` such as `Buttons`, `EditTexts` and `RadioButtons`. This means that all of these views support the same functionality. Insert an `EditText` into the XML file. Use the `android:drawableLeft`, `android:hint` and the `android:drawablePadding` attributes to insert some hint text to the user and the android icon to the left of the `EditText` (figure 11).

```
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Username"
    android:drawableLeft="@mipmap/ic_launcher"
    android:drawablePadding="8dp"
    />
```



Working with XML Properties

This is sans
This is serif
This is monospace
This is normal
A light blue colour
My Contacts
Username

FIGURE 11 – IMAGES WITHIN AN EDITTEXT VIEW

EXERCISE 3 - Working with the EditText

The EditText is the standard text entry widget in Android apps. If the user needs to enter text into an app, this is the primary way for them to do that (figure 12).

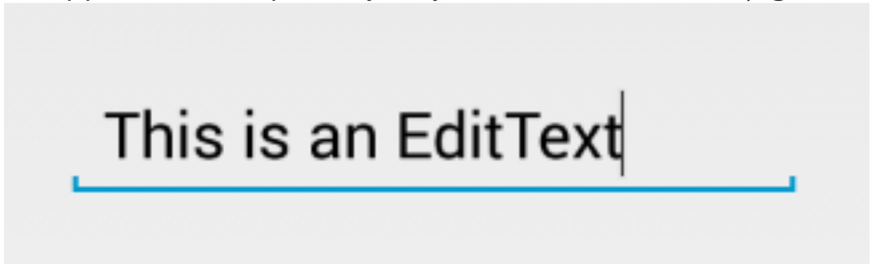


FIGURE 12 - EDITTEXT

There are many important attributes that can be set to customize the behavior of an EditText.

Task 1 - Add an EditText

Add an EditText to the XML file and give it an ID (figure 13)

```
<EditText
    android:id="@+id/et_simple"
    android:layout_height="wrap_content"
    android:layout_width="match_parent"
/>
```

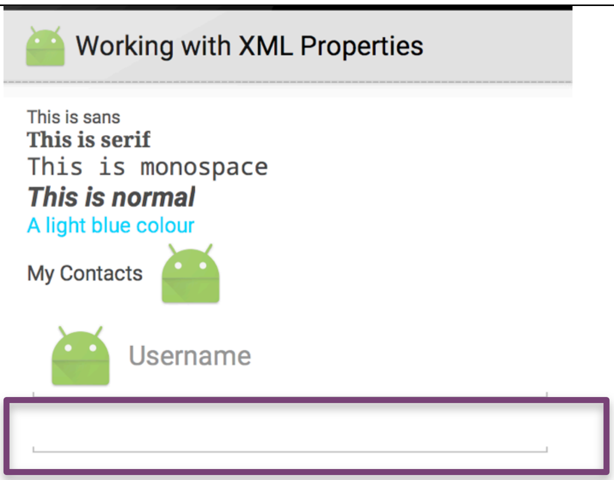


FIGURE 13 – AN EDITTEXT VIEW

Note that an EditText is simply a thin extension of the TextView and inherits all of the same properties.

Task 2 - Customizing the Input Type

By default, any text contained within an `EditText` control is displayed as plain text. By setting `inputType`, we can facilitate input of different types of information, like phone numbers and passwords:

```
<EditText
    android:id="@+id/et_simple"
    android:layout_height="wrap_content"
    android:layout_width="match_parent"
    android:inputType="number"
/>
```

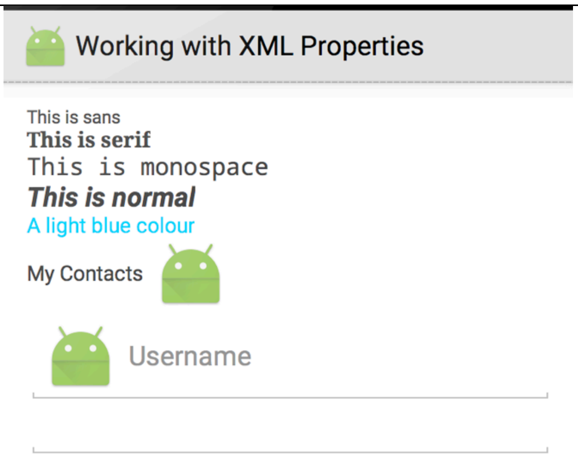


Figure 14 – Setting the InputType

Most common input types include:

Type	Description
`textUri`	Text that will be used as a URI
`textEmailAddress`	Text that will be used as an e-mail address
`textPersonName`	Text that is the name of a person
`textPassword`	Text that is a password that should be obscured
`number`	A numeric only field
`phone`	For entering a phone number
`date`	For entering a date
`time`	For entering a time
`textMultiLine`	Allow multiple lines of text in the field

Set multiple `inputType` attributes if needed (separated by '|')

```
<EditText
    android:id="@+id/et_simple"
    android:layout_height="wrap_content"
    android:layout_width="match_parent"
    android:inputType="date|datetime"
/>
```

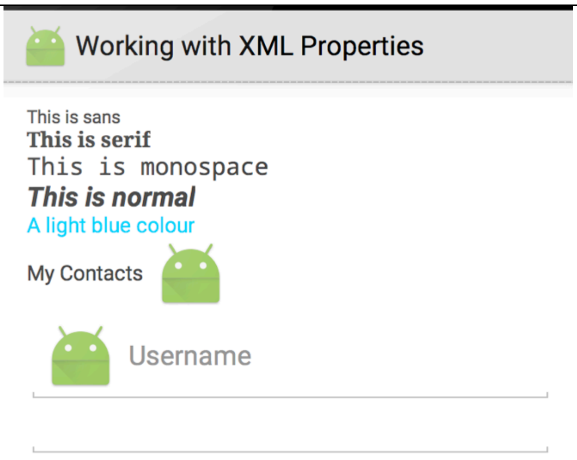


FIGURE 15 – SETTING THE INPUTTYPE

Limit the entry to a single-line of text (avoid newlines)

```
<EditText
  android:singleLine="true"
  android:lines="1"
/>
```

Limit the characters that can be entered into a field using the digits attribute. This would restrict the digits entered to just "0" and "1".

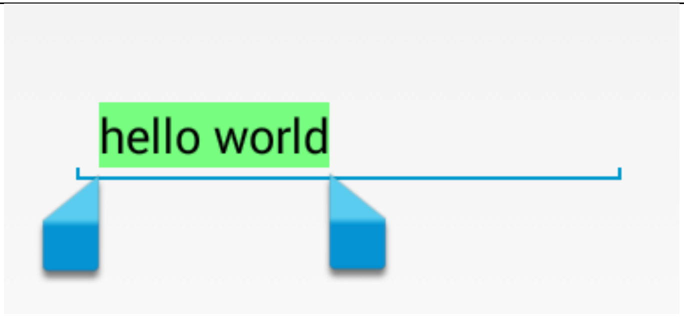
```
<EditText
  android:inputType="number"
  android:digits="01"
/>
```

Limit the total number of characters with

```
<EditText
  android:maxLength="5"
/>
```

Task 3 - Adjusting Colors

Adjust the highlight background color of selected text within an `EditText` with the `android:textColorHighlight` attribute.

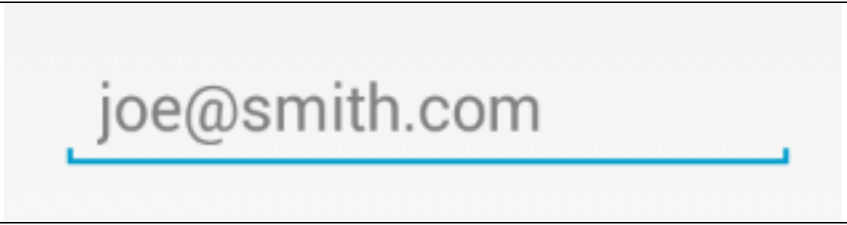
`<EditText android:textColorHighlight="#7cff88" />`	 FIGURE 16 - ADJUSTING COLOUR

```
<EditText
    android:textColorHighlight="#7cff88"
/>
```

Task 4 - Displaying Placeholder Hints

Set the hint for the EditText control to prompt a user for specific input

```
<EditText
    ...
    android:hint="joe@smith.com">
</EditText>
```


joe@smith.com

EXERCISE 4- Working with the `ImageView`

Typically, images are displayed using the built-in image view. This view takes care of the loading and optimizing of the image, freeing you to focus on app-specific details like the layout and content.

Task 1 – Add an `ImageView`

Add an image called phone.png to your `drawable` folder.

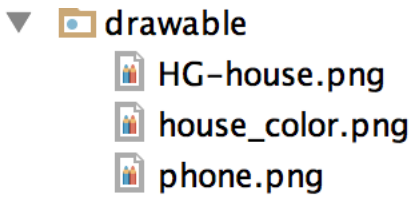
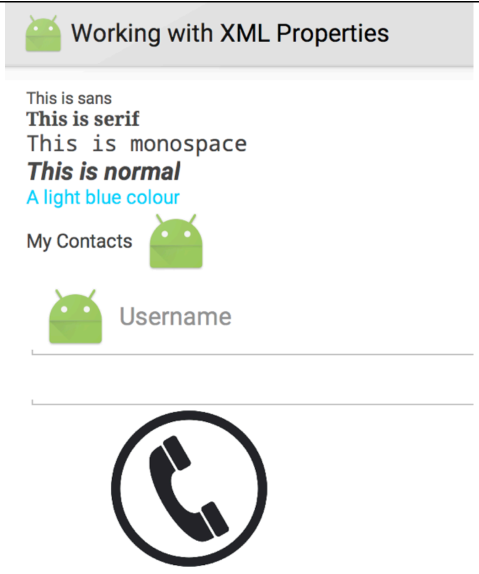


FIGURE 17 - ADD AN IMAGE TO THE DRAWABLE FOLDER

At the simplest level, an `ImageView` is simply a view you embed within an XML layout that is used to display an image (or any drawable) on the screen. The `ImageView` looks like this in `res/layout/activity_main.xml`

```<ImageView android:id="@+id/image" android:layout_width="238dp" android:layout_height="114dp" android:scaleType="fitCenter" android:src="@drawable/phone" />```	 FIGURE 18- ADDING A PICTURE TO THE CANVAS

```
<ImageView
    android:id="@+id/image"
    android:layout_width="238dp"
    android:layout_height="114dp"
    android:scaleType="fitCenter"
    android:src="@drawable/phone" />
```

The `ImageView` handles all the loading and scaling of the image for you. Note the `scaleType` attribute, which defines how the images will be scaled to fit in your layout. In the example, using `scaleType="fitCenter"`, the image will be displayed at fitted and centered in the view.

Task 2 - Sizing `ImageView` Controls

By default, contents of an `ImageView` control are of a certain size- usually the size of the image dimensions. They can also be bounded by their `layout_width` and `layout_height` attributes.

```
<ImageView
    android:layout_width="50dp"
    android:layout_height="50dp"
    android:scaleType="fitXY"
    ...
/>
```

The `scaleType` above has been set to `fitXY`, which sets the height, and the width up or down to fit the maximum dimensions specified.

Fixing the width and height however means that the proportions of the width and height of the original image, known as the aspect ratio, will be altered. Take advantage of the `adjustViewBounds` parameter to preserve this aspect ratio. However, either allow the height and/or width to be adjustable (i.e. by using `maxWidth` and using `wrap_content` for the dimension). Otherwise, the dimensions cannot be readjusted to meet the required aspect ratio.

```
<ImageView
    android:layout_width="50dp"
    android:layout_height="wrap_content"
    android:scaleType="fitXY"
    android:adjustViewBounds="true"
/>
```

Scale Types

An `ImageView` can display an image differently based on the `scaleType` provided. The following is a list of all the most common types:

Scale Type	Description
`center`	Displays the image centered in the view with no scaling.
`centerCrop`	Scales the image such that both the x and y dimensions are greater than or equal to the view, while maintaining the image aspect ratio; centers the image in the view.
`centerInside`	Scales the image to fit inside the view, while maintaining the image aspect ratio. If the image is already smaller than the view, then this is the same as center.
`fitCenter`	Scales the image to fit inside the view, while maintaining the image aspect ratio. At least one axis will exactly match the view, and the result is centered inside the view.
`fitStart`	Same as fitCenter but aligned to the top left of the view.
`fitEnd`	Same as fitCenter but aligned to the bottom right of the view.
`fitXY`	Scales the x and y dimensions to exactly match the view size; does not maintain the image aspect ratio.
`matrix`	Scales the image using a supplied Matrix class. The matrix can be supplied using the setImageMatrix method. A Matrix class can be used to apply transformations such as rotations to an image.

Note on `Mipmaps` and `Drawables`

- Starting with Android 4.3, there is now an option to use the `res/mipmap` folder to store `"mipmap"` images.
- `Mipmaps` are most **commonly used for application icons** such as the launcher icon.
- `Mipmap` image resources can then be accessed using the `@mipmap/ic_launcher` notation in place of `@drawable`.
- Placing icons in `mipmap` folders (rather than `drawable`) is considered a best practice because they can often be used at resolutions different from the device's current density.