# **AppCompatActivity**

**androidx.appcompat.app.AppCompatActivity. Base class for activities that wish to use some of the newer platform features on older Android devices. Some of these backported features include: Using the action bar, including action items, navigation modes and more with the setSupportActionBar API.**

# **Bundles**

**Android Bundles are generally used for passing data from one activity to another. Basically here the concept of key-value pair is used where the data that one wants to pass is the value of the map, which can be later retrieved by using the key.**

# **SavedInstanceState**

**The savedInstanceState is a reference to a Bundle object that is passed into the onCreate method of every Android Activity.**

# **Android | build.gradle**

Gradle is a build system (open source) that is used to automate building, testing, deployment, etc. “build.gradle” are scripts where one can automate the tasks. For example, the simple task to copy some files from one directory to another can be performed by the Gradle build script before the actual build process happens.

### **Why is Gradle Needed?**

Every Android project needs a Gradle for generating an apk from the *.java* and *.xml* files in the project. Simply put, a gradle takes all the source files (java and XML) and applies appropriate tools, e.g., converts the java files into dex files and compresses all of them into a single file known as apk that is actually used. There are two types of build.gradle scripts

* Top-level build.gradle
* Module-level build.gradle

**Top-level build.gradle:**

It is located in the root project directory and its main function is to define the build configurations that will be applied to all the modules in the project. It is implemented as:

***buildscript:*** This block is used to configure the repositories and dependencies for Gradle.

***Note:*** *Don’t include dependencies here. (those will be included in the module-level build.gradle)*

***dependencies:*** This block in build script is used to configure dependencies that the Gradle needs to build during the project.

**Anatomy Of An Android Application**

**OR**

**Android Terminologies & Resource handling**

### **1. Activities**

Activities are said to be the presentation layer of our applications. The UI of our application is built around one or more extensions of the Activity class. By using Fragments and Views, activities set the layout and display the output and also respond to the user’s actions. An activity is implemented as a subclass of class Activity.

| public class MainActivity extends Activity {  } |
| --- |

### **2. Services**

Services are like invisible workers of our app. These components run at the backend, updating your data sources and Activities, triggering Notification, and also broadcast Intents. They also perform some tasks when applications are not active. A service can be used as a subclass of class Service:

| **public class ServiceName extends Service {**  **}** |
| --- |

### **3. Content Providers**

It is used to manage and persist the application data also typically interacts with the SQL database. They are also responsible for sharing the data beyond the application boundaries. The Content Providers of a particular application can be configured to allow access from other applications, and the Content Providers exposed by other applications can also be configured.

A content provider should be a sub-class of the class ContentProvider.

| **public class contentProviderName extends ContentProvider {**  **public void onCreate(){}**  **}** |
| --- |

### **4. Broadcast Receivers**

They are known to be intent listeners as they enable your application to listen to the Intents that satisfy the matching criteria specified by us. Broadcast Receivers make our application react to any received Intent thereby making them perfect for creating event-driven applications.

### **5. Intents**

It is a powerful inter-application message-passing framework. They are extensively used throughout Android. Intents can be used to start and stop Activities and Services, to broadcast messages system-wide or to an explicit Activity, Service or Broadcast Receiver or to request action be performed on a particular piece of data.

### **6. Widgets**

These are the small visual application components that you can find on the home screen of the devices. They are a special variation of Broadcast Receiver that allow us to create dynamic, interactive application components for users to embed on their Home Screen.

### **7. Notifications**

Notifications are the application alerts that are used to draw the user’s attention to some particular app event without stealing focus or interrupting the current activity of the user. They are generally used to grab user’s attention when the application is not visible or active, particularly from within a Service or Broadcast Receiver. Examples: Email popups, Messenger popups, etc.

#### **8 . Fragment**

Fragments are like parts of activity. An activity can display one or more fragments on the screen at the same time.

#### **9 . AndroidManifest.xml**

It contains information about activities, content providers, permissions etc. It is like the web.xml file in Java EE.

**Working with Different Types of Resources**

String resources

A string resource provides text strings for your application with optional text styling and formatting. There are three types of resources that can provide your application with strings:

**String**

XML resource that provides a single string.

**String Array**

XML resource that provides an array of strings.

**Quantity Strings**

XML resource that carries different strings for pluralization.

All strings are capable of applying some styling markup and formatting arguments. For information about styling and formatting strings, see the section about formatting and styling.

| **String** A single string that can be referenced from the application or from other resource files (such as an XML layout).  **Note:** A string is a simple resource that is referenced using the value provided in the **name** attribute (not the name of the XML file). So, you can combine string resources with other simple resources in the one XML file, under one **<resources>** element.  file location:  res/values/*filename*.xml  The filename is arbitrary. The <string> element's name is used as the resource ID.  compiled resource datatype:  Resource pointer to a String.  resource reference:  In Java: R.string.*string\_name*  In XML:@string/*string\_name*  syntax:  <?xml version="1.0" encoding="utf-8"?>  <resource>  <string  name="*string\_name*"  >*text\_string*</string>  </resources>  elements:  <resources>  **Required.** This must be the root node.  No attributes.  <string>  A string, which can include styling tags. Beware that you must escape apostrophes and quotation marks. For more information about how to properly style and format your strings see Formatting and Styling below:  **attributes:**  name  *String*. A name for the string. This name is used as the resource ID.  example:  XML file saved at res/values/strings.xml:  <?xml version="1.0" encoding="utf-8"?>  <resources>  <string name="hello">Hello!</string>  </resources>  This layout XML applies a string to a View:  <TextView  android:layout\_width="fill\_parent"  android:layout\_height="wrap\_content"  **android:text="@string/hello"** />  This application code retrieves a string:  String string = getString(R.string.hello);  You can use either [getString(int)](https://developer.android.com/reference/android/content/Context#getString(int)) or [getText(int)](https://developer.android.com/reference/android/content/Context#getText(int)) to retrieve a string. [getText(int)](https://developer.android.com/reference/android/content/Context#getText(int)) retains any rich text styling applied to the string. |
| --- |

| **String array** An array of strings that can be referenced from the application.  **Note:** A string array is a simple resource that is referenced using the value provided in the **name** attribute (not the name of the XML file). As such, you can combine string array resources with other simple resources in the one XML file, under one **<resources>** element.  file location:  res/values/*filename*.xml  The filename is arbitrary. The <string-array> element's name is used as the resource ID.  compiled resource datatype:  Resource pointer to an array of [String](https://developer.android.com/reference/java/lang/String)s.  resource reference:  In Java: R.array.*string\_array\_name*  In XML: @[*package*:]array/*string\_array\_name*  syntax:  <?xml version="1.0" encoding="utf-8"?>  <[**resources**](https://developer.android.com/guide/topics/resources/string-resource#string-array-resources-element)>  <[**string-array**](https://developer.android.com/guide/topics/resources/string-resource#string-array-element)  name="*string\_array\_name*">  <[**item**](https://developer.android.com/guide/topics/resources/string-resource#string-array-item-element)  >*text\_string*</item>  </string-array>  </resources>  elements:  <resources>  **Required.** This must be the root node.  No attributes.  <string-array>  Defines an array of strings. Contains one or more <item> elements.  attributes:  name  *String*. A name for the array. This name is used as the resource ID to reference the array.  <item>  A string, which can include styling tags. The value can be a reference to another string resource. Must be a child of a <string-array> element. Beware that you must escape apostrophes and quotation marks. See Formatting and Style below, for information about properly style and format your strings.  No attributes.  **example:**  **XML file saved at res/values/strings.xml:**  <?xml version="1.0" encoding="utf-8"?>  <resources>  <string-array name="planets\_array">  <item>Mercury</item>  <item>Venus</item>  <item>Earth</item>  <item>Mars</item>  </string-array>  </resources>  **This application code retrieves a**  Resources res = [**getResources()**](https://developer.android.com/reference/android/content/Context#getResources());  String[] planets = res.[**getStringArray**](https://developer.android.com/reference/android/content/res/Resources#getStringArray(int))(R.array.planets\_array); |
| --- |

More resource types

This page defines more types of resources you can externalize, including:

[Bool](https://developer.android.com/guide/topics/resources/more-resources#Bool)

XML resource that carries a boolean value.

[Color](https://developer.android.com/guide/topics/resources/more-resources#Color)

XML resource that carries a color value (a hexadecimal color).

[Dimension](https://developer.android.com/guide/topics/resources/more-resources#Dimension)

XML resource that carries a dimension value (with a unit of measure).

[ID](https://developer.android.com/guide/topics/resources/more-resources#Id)

XML resource that provides a unique identifier for application resources and components.

[Integer](https://developer.android.com/guide/topics/resources/more-resources#Integer)

XML resource that carries an integer value.

[Integer Array](https://developer.android.com/guide/topics/resources/more-resources#IntegerArray)

XML resource that provides an array of integers.

[Typed Array](https://developer.android.com/guide/topics/resources/more-resources#TypedArray)

XML resource that provides a [TypedArray](https://developer.android.com/reference/android/content/res/TypedArray) (which you can use for an array of drawables).

| **Bool** A boolean value defined in XML.  **Note:** A bool is a simple resource that is referenced using the value provided in the **name** attribute (not the name of the XML file). As such, you can combine bool resources with other simple resources in the one XML file, under one **<resources>** element.  file location:  res/values/*filename*.xml  The filename is arbitrary. The <bool> element's name will be used as the resource ID.  resource reference:  In Java: R.bool.*bool\_name*  In XML: @[*package*:]bool/*bool\_name*  syntax:  <?xml version="1.0" encoding="utf-8"?>  <[**resources**](https://developer.android.com/guide/topics/resources/more-resources#bool-resources-element)>  <[**bool**](https://developer.android.com/guide/topics/resources/more-resources#bool-element)  name="*bool\_name*"  >[true | false]</bool>  </resources>  elements:  <resources>  **Required.** This must be the root node.  No attributes.  <bool>  A boolean value: true or false.  attributes:  name  *String*. A name for the bool value. This will be used as the resource ID.  example:  XML file saved at res/values-small/bools.xml:  <?xml version="1.0" encoding="utf-8"?>  <resources>  <bool name="screen\_small">true</bool>  <bool name="adjust\_view\_bounds">true</bool>  </resources>  This application code retrieves the boolean:  Resources res = [**getResources()**](https://developer.android.com/reference/android/content/Context#getResources());  boolean screenIsSmall = res.[**getBoolean**](https://developer.android.com/reference/android/content/res/Resources#getBoolean(int))(R.bool.screen\_small);  This layout XML uses the boolean for an attribute:  <ImageView  android:layout\_height="fill\_parent"  android:layout\_width="fill\_parent"  android:src="@drawable/logo"  android:adjustViewBounds="@bool/adjust\_view\_bounds" /> |
| --- |

| **Color** A color value defined in XML. The color is specified with an RGB value and alpha channel. You can use a color resource any place that accepts a hexadecimal color value. You can also use a color resource when a drawable resource is expected in XML (for example, android:drawable="@color/green").  The value always begins with a pound (#) character and then followed by the Alpha-Red-Green-Blue information in one of the following formats:   * #*RGB* * #*ARGB* * #*RRGGBB* * #*AARRGGBB*   **Note:** A color is a simple resource that is referenced using the value provided in the **name** attribute (not the name of the XML file). As such, you can combine color resources with other simple resources in the one XML file, under one **<resources>** element.  file location:  res/values/colors.xml  The filename is arbitrary. The <color> element's name will be used as the resource ID.  resource reference:  In Java: R.color.*color\_name*  In XML: @[*package*:]color/*color\_name*  syntax:  <?xml version="1.0" encoding="utf-8"?>  <[**resources**](https://developer.android.com/guide/topics/resources/more-resources#color-resources-element)>  <[**color**](https://developer.android.com/guide/topics/resources/more-resources#color-element)  name="*color\_name*"  >*hex\_color*</color>  </resources>  elements:  <resources>  **Required.** This must be the root node.  No attributes.  <color>  A color expressed in hexadecimal, as described above.  attributes:  name  *String*. A name for the color. This will be used as the resource ID.  example:  XML file saved at res/values/colors.xml:  <?xml version="1.0" encoding="utf-8"?>  <resources>  <color name="opaque\_red">#f00</color>  <color name="translucent\_red">#80ff0000</color>  </resources>  This application code retrieves the color resource:  Resources res = [**getResources()**](https://developer.android.com/reference/android/content/Context#getResources());  int color = res.[**getColor**](https://developer.android.com/reference/android/content/res/Resources#getColor(int))(R.color.opaque\_red);  This layout XML applies the color to an attribute:  <TextView  android:layout\_width="fill\_parent"  android:layout\_height="wrap\_content"  android:textColor="@color/translucent\_red"  android:text="Hello"/> |
| --- |

| **Dimension** A dimension value defined in XML. A dimension is specified with a number followed by a unit of measure. For example: 10px, 2in, 5sp. The following units of measure are supported by Android:  dp  Density-independent Pixels - An abstract unit that is based on the physical density of the screen. These units are relative to a 160 dpi (dots per inch) screen, on which 1dp is roughly equal to 1px. When running on a higher density screen, the number of pixels used to draw 1dp is scaled up by a factor appropriate for the screen's dpi. Likewise, when on a lower density screen, the number of pixels used for 1dp is scaled down. The ratio of dp-to-pixel will change with the screen density, but not necessarily in direct proportion. Using dp units (instead of px units) is a simple solution to making the view dimensions in your layout resize properly for different screen densities. In other words, it provides consistency for the real-world sizes of your UI elements across different devices.  sp  Scale-independent Pixels - This is like the dp unit, but it is also scaled by the user's font size preference. It is recommend you use this unit when specifying font sizes, so they will be adjusted for both the screen density and the user's preference.  pt  Points - 1/72 of an inch based on the physical size of the screen, assuming a 72dpi density screen.  px  Pixels - Corresponds to actual pixels on the screen. This unit of measure is not recommended because the actual representation can vary across devices; different devices may have a different number of pixels per inch and may have more or fewer total pixels available on the screen.  mm  Millimeters - Based on the physical size of the screen.  in  Inches - Based on the physical size of the screen.  **Note:** A dimension is a simple resource that is referenced using the value provided in the **name** attribute (not the name of the XML file). As such, you can combine dimension resources with other simple resources in the one XML file, under one **<resources>** element.  file location:  res/values/*filename*.xml  The filename is arbitrary. The <dimen> element's name will be used as the resource ID.  resource reference:  In Java: R.dimen.*dimension\_name*  In XML: @[*package*:]dimen/*dimension\_name*  syntax:  <?xml version="1.0" encoding="utf-8"?>  <[**resources**](https://developer.android.com/guide/topics/resources/more-resources#dimen-resources-element)>  <[**dimen**](https://developer.android.com/guide/topics/resources/more-resources#dimen-element)  name="*dimension\_name*"  >*dimension*</dimen>  </resources>  elements:  <resources>  **Required.** This must be the root node.  No attributes.  <dimen>  A dimension, represented by a float, followed by a unit of measurement (dp, sp, pt, px, mm, in), as described above.  attributes:  name  *String*. A name for the dimension. This will be used as the resource ID.  example:  XML file saved at res/values/dimens.xml:  <?xml version="1.0" encoding="utf-8"?>  <resources>  <dimen name="textview\_height">25dp</dimen>  <dimen name="textview\_width">150dp</dimen>  <dimen name="ball\_radius">30dp</dimen>  <dimen name="font\_size">16sp</dimen>  </resources>  This application code retrieves a dimension:  Resources res = [**getResources()**](https://developer.android.com/reference/android/content/Context#getResources());  float fontSize = res.[**getDimension**](https://developer.android.com/reference/android/content/res/Resources#getDimension(int))(R.dimen.font\_size);  This layout XML applies dimensions to attributes:  <TextView  android:layout\_height="@dimen/textview\_height"  android:layout\_width="@dimen/textview\_width"  android:textSize="@dimen/font\_size"/> |
| --- |

| **ID** A unique resource ID defined in XML. Using the name you provide in the <item> element, the Android developer tools create a unique integer in your project's R.java class, which you can use as an identifier for an application resources (for example, a [View](https://developer.android.com/reference/android/view/View) in your UI layout) or a unique integer for use in your application code (for example, as an ID for a dialog or a result code).  **Note:** An ID is a simple resource that is referenced using the value provided in the **name** attribute (not the name of the XML file). As such, you can combine ID resources with other simple resources in the one XML file, under one **<resources>** element. Also, remember that an ID resources does not reference an actual resource item; it is simply a unique ID that you can attach to other resources or use as a unique integer in your application.  file location:  res/values/*filename.xml*  The filename is arbitrary.  resource reference:  In Java: R.id.*name*  In XML: @[*package*:]id/*name*  syntax:  <?xml version="1.0" encoding="utf-8"?>  <[**resources**](https://developer.android.com/guide/topics/resources/more-resources#id-resources-element)>  <[**item**](https://developer.android.com/guide/topics/resources/more-resources#id-item-element)  type="id"  name="*id\_name*" />  </resources>  elements:  <resources>  **Required.** This must be the root node.  No attributes.  <item>  Defines a unique ID. Takes no value, only attributes.  attributes:  type  Must be "id".  name  *String*. A unique name for the ID.  example:  XML file saved at res/values/ids.xml:  <?xml version="1.0" encoding="utf-8"?>  <resources>  <item type="id" name="button\_ok" />  <item type="id" name="dialog\_exit" />  </resources>  Then, this layout snippet uses the "button\_ok" ID for a Button widget:  <Button android:id="**@id/button\_ok**"  style="@style/button\_style" />  Notice that the android:id value does not include the plus sign in the ID reference, because the ID already exists, as defined in the ids.xml example above. (When you specify an ID to an XML resource using the plus sign—in the format android:id="@+id/name"—it means that the "name" ID does not exist and should be created.)  As another example, the following code snippet uses the "dialog\_exit" ID as a unique identifier for a dialog:  [showDialog](https://developer.android.com/reference/android/app/Activity#showDialog(int))(R.id.dialog\_exit);  In the same application, the "dialog\_exit" ID is compared when creating a dialog:  protected Dialog [onCreateDialog](https://developer.android.com/reference/android/app/Activity#onCreateDialog(int))(int id) {  Dialog dialog;  switch(id) {  case R.id.dialog\_exit:  ...  break;  default:  dialog = null;  }  return dialog;  } |
| --- |

| **Integer** An integer defined in XML.  **Note:** An integer is a simple resource that is referenced using the value provided in the **name** attribute (not the name of the XML file). As such, you can combine integer resources with other simple resources in the one XML file, under one **<resources>** element.  file location:  res/values/*filename.xml*  The filename is arbitrary. The <integer> element's name will be used as the resource ID.  resource reference:  In Java: R.integer.*integer\_name*  In XML: @[*package*:]integer/*integer\_name*  syntax:  <?xml version="1.0" encoding="utf-8"?>  <[**resources**](https://developer.android.com/guide/topics/resources/more-resources#integer-resources-element)>  <[**integer**](https://developer.android.com/guide/topics/resources/more-resources#integer-element)  name="*integer\_name*"  >*integer*</integer>  </resources>  elements:  <resources>  **Required.** This must be the root node.  No attributes.  <integer>  An integer.  attributes:  name  *String*. A name for the integer. This will be used as the resource ID.  example:  XML file saved at res/values/integers.xml:  <?xml version="1.0" encoding="utf-8"?>  <resources>  <integer name="max\_speed">75</integer>  <integer name="min\_speed">5</integer>  </resources>  This application code retrieves an integer:  Resources res = [**getResources()**](https://developer.android.com/reference/android/content/Context#getResources());  int maxSpeed = res.[**getInteger**](https://developer.android.com/reference/android/content/res/Resources#getInteger(int))(R.integer.max\_speed); |
| --- |

| **Integer array** An array of integers defined in XML.  **Note:** An integer array is a simple resource that is referenced using the value provided in the **name** attribute (not the name of the XML file). As such, you can combine integer array resources with other simple resources in the one XML file, under one **<resources>** element.  file location:  res/values/*filename*.xml  The filename is arbitrary. The <integer-array> element's name will be used as the resource ID.  compiled resource datatype:  Resource pointer to an array of integers.  resource reference:  In Java: R.array.*integer\_array\_name*  In XML: @[*package*:]array/*integer\_array\_name*  syntax:  <?xml version="1.0" encoding="utf-8"?>  <[**resources**](https://developer.android.com/guide/topics/resources/more-resources#integer-array-resources-element)>  <[**integer-array**](https://developer.android.com/guide/topics/resources/more-resources#integer-array-element)  name="*integer\_array\_name*">  <[**item**](https://developer.android.com/guide/topics/resources/more-resources#integer-array-item-element)  >*integer*</item>  </integer-array>  </resources>  elements:  <resources>  **Required.** This must be the root node.  No attributes.  <integer-array>  Defines an array of integers. Contains one or more child <item> elements.  attributes:  android:name  *String*. A name for the array. This name will be used as the resource ID to reference the array.  <item>  An integer. The value can be a reference to another integer resource. Must be a child of a <integer-array> element.  No attributes.  example:  XML file saved at res/values/integers.xml:  <?xml version="1.0" encoding="utf-8"?>  <resources>  <integer-array name="bits">  <item>4</item>  <item>8</item>  <item>16</item>  <item>32</item>  </integer-array>  </resources>  This application code retrieves the integer array:  Resources res = [**getResources()**](https://developer.android.com/reference/android/content/Context#getResources());  int[] bits = res.[**getIntArray**](https://developer.android.com/reference/android/content/res/Resources#getIntArray(int))(R.array.bits); |
| --- |

| **Typed array** A [TypedArray](https://developer.android.com/reference/android/content/res/TypedArray) defined in XML. You can use this to create an array of other resources, such as drawables. Note that the array is not required to be homogeneous, so you can create an array of mixed resource types, but you must be aware of what and where the data types are in the array so that you can properly obtain each item with the [TypedArray](https://developer.android.com/reference/android/content/res/TypedArray)'s get...() methods.  **Note:** A typed array is a simple resource that is referenced using the value provided in the **name** attribute (not the name of the XML file). As such, you can combine typed array resources with other simple resources in the one XML file, under one **<resources>** element.  file location:  res/values/*filename*.xml  The filename is arbitrary. The <array> element's name will be used as the resource ID.  compiled resource datatype:  Resource pointer to a [TypedArray](https://developer.android.com/reference/android/content/res/TypedArray).  resource reference:  In Java: R.array.*array\_name*  In XML: @[*package*:]array/*array\_name*  syntax:  <?xml version="1.0" encoding="utf-8"?>  <[**resources**](https://developer.android.com/guide/topics/resources/more-resources#array-resources-element)>  <[**array**](https://developer.android.com/guide/topics/resources/more-resources#array-element)  name="*integer\_array\_name*">  <[**item**](https://developer.android.com/guide/topics/resources/more-resources#array-item-element)>*resource*</item>  </array>  </resources>  elements:  <resources>  **Required.** This must be the root node.  No attributes.  <array>  Defines an array. Contains one or more child <item> elements.  attributes:  android:name  *String*. A name for the array. This name will be used as the resource ID to reference the array.  <item>  A generic resource. The value can be a reference to a resource or a simple data type. Must be a child of an <array> element.  No attributes.  example:  XML file saved at res/values/arrays.xml:  <?xml version="1.0" encoding="utf-8"?>  <resources>  <array name="icons">  <item>@drawable/home</item>  <item>@drawable/settings</item>  <item>@drawable/logout</item>  </array>  <array name="colors">  <item>#FFFF0000</item>  <item>#FF00FF00</item>  <item>#FF0000FF</item>  </array>  </resources>  This application code retrieves each array and then obtains the first entry in each array:  Resources res = [**getResources()**](https://developer.android.com/reference/android/content/Context#getResources());  TypedArray icons = res.[**obtainTypedArray**](https://developer.android.com/reference/android/content/res/Resources#obtainTypedArray(int))(R.array.icons);  Drawable drawable = icons.[**getDrawable**](https://developer.android.com/reference/android/content/res/TypedArray#getDrawable(int))(0);  TypedArray colors = res.[**obtainTypedArray**](https://developer.android.com/reference/android/content/res/Resources#obtainTypedArray(int))(R.array.colors);  int color = colors.[**getColor**](https://developer.android.com/reference/android/content/res/TypedArray#getColor(int,%20int))(0,0); |
| --- |

Animation

| An animation resource can define one of two types of animations:  [Property Animation](https://developer.android.com/guide/topics/resources/animation-resource#Property)  Creates an animation by modifying an object's property values over a set period of time with an [Animator](https://developer.android.com/reference/android/animation/Animator).  [View Animation](https://developer.android.com/guide/topics/resources/animation-resource#View)  There are two types of animations that you can do with the view animation framework:   * [Tween animation](https://developer.android.com/guide/topics/resources/animation-resource#Tween): Creates an animation by performing a series of transformations on a single image with an [Animation](https://developer.android.com/reference/android/view/animation/Animation) * [Frame animation](https://developer.android.com/guide/topics/resources/animation-resource#Frame): or creates an animation by showing a sequence of images in order with an [AnimationDrawable](https://developer.android.com/reference/android/graphics/drawable/AnimationDrawable).  **Property animation** An animation defined in XML that modifies properties of the target object, such as background color or alpha value, over a set amount of time.  file location:  res/animator/*filename*.xml  The filename will be used as the resource ID.  compiled resource datatype:  Resource pointer to a [ValueAnimator](https://developer.android.com/reference/android/animation/ValueAnimator), [ObjectAnimator](https://developer.android.com/reference/android/animation/ObjectAnimator), or [AnimatorSet](https://developer.android.com/reference/android/animation/AnimatorSet).  resource reference:  In Java-based or Kotlin code: R.animator.*filename*  In XML: @[*package*:]animator/*filename*  syntax:  <[**set**](https://developer.android.com/guide/topics/resources/animation-resource#animator-set-element)  android:ordering=["together" | "sequentially"]>  <[**objectAnimator**](https://developer.android.com/guide/topics/resources/animation-resource#obj-animator-element)  android:propertyName="*string*"  android:duration="*int*"  android:valueFrom="*float* | *int* | *color*"  android:valueTo="*float* | *int* | *color*"  android:startOffset="*int*"  android:repeatCount="*int*"  android:repeatMode=["restart" | "reverse"]  android:valueType=["intType" | "floatType"]/>  <[**animator**](https://developer.android.com/guide/topics/resources/animation-resource#val-animator-element)  android:duration="*int*"  android:valueFrom="*float* | *int* | *color*"  android:valueTo="*float* | *int* | *color*"  android:startOffset="*int*"  android:repeatCount="*int*"  android:repeatMode=["restart" | "reverse"]  android:valueType=["intType" | "floatType"]/>  <[**set**](https://developer.android.com/guide/topics/resources/animation-resource#animator-set-element)>  ...  </set>  </set>  The file must have a single root element: either <set>, <objectAnimator>, or <valueAnimator>. You can group animation elements together inside the <set> element, including other <set> elements.  elements:  <set>  A container that holds other animation elements (<objectAnimator>, <valueAnimator>, or other <set> elements). Represents an [AnimatorSet](https://developer.android.com/reference/android/animation/AnimatorSet).  You can specify nested <set> tags to further group animations together. Each <set> can define its own ordering attribute.  attributes:  android:ordering  *Keyword*. Specifies the play ordering of animations in this set.   | **Value** | **Description** | | --- | --- | | sequentially | Play animations in this set sequentially | | together (default) | Play animations in this set at the same time. |   <objectAnimator>  Animates a specific property of an object over a specific amount of time. Represents an [ObjectAnimator](https://developer.android.com/reference/android/animation/ObjectAnimator).  attributes:  android:propertyName  *String*. **Required**. The object's property to animate, referenced by its name. For example you can specify "alpha" or "backgroundColor" for a View object. The objectAnimator element does not expose a target attribute, however, so you cannot set the object to animate in the XML declaration. You have to inflate your animation XML resource by calling [loadAnimator()](https://developer.android.com/reference/android/animation/AnimatorInflater#loadAnimator(android.content.Context,%20int)) and call [setTarget()](https://developer.android.com/reference/android/animation/ObjectAnimator#setTarget(java.lang.Object)) to set the target object that contains this property.  android:valueTo  *float, int, or color*. **Required**. The value where the animated property ends. Colors are represented as six digit hexadecimal numbers (for example, #333333).  android:valueFrom  *float, int, or color*. The value where the animated property starts. If not specified, the animation starts at the value obtained by the property's get method. Colors are represented as six digit hexadecimal numbers (for example, #333333).  android:duration  *int*. The time in milliseconds of the animation. 300 milliseconds is the default.  android:startOffset  *int*. The amount of milliseconds the animation delays after [start()](https://developer.android.com/reference/android/animation/ObjectAnimator#start()) is called.  android:repeatCount  *int*. How many times to repeat an animation. Set to "-1" to infinitely repeat or to a positive integer. For example, a value of "1" means that the animation is repeated once after the initial run of the animation, so the animation plays a total of two times. The default value is "0", which means no repetition.  android:repeatMode  *int*. How an animation behaves when it reaches the end of the animation. android:repeatCount must be set to a positive integer or "-1" for this attribute to have an effect. Set to "reverse" to have the animation reverse direction with each iteration or "restart" to have the animation loop from the beginning each time.  android:valueType  *Keyword*. Do not specify this attribute if the value is a color. The animation framework automatically handles color values   | **Value** | **Description** | | --- | --- | | intType | Specifies that the animated values are integers | | floatType (default) | Specifies that the animated values are floats |   <animator>  Performs an animation over a specified amount of time. Represents a [ValueAnimator](https://developer.android.com/reference/android/animation/ValueAnimator).  attributes:  android:valueTo  *float, int, or color*. **Required**. The value where the animation ends. Colors are represented as six digit hexadecimal numbers (for example, #333333).  android:valueFrom  *float, int, or color*. **Required**. The value where the animation starts. Colors are represented as six digit hexadecimal numbers (for example, #333333).  android:duration  *int*. The time in milliseconds of the animation. 300ms is the default.  android:startOffset  *int*. The amount of milliseconds the animation delays after [start()](https://developer.android.com/reference/android/animation/ValueAnimator#start()) is called.  android:repeatCount  *int*. How many times to repeat an animation. Set to "-1" to infinitely repeat or to a positive integer. For example, a value of "1" means that the animation is repeated once after the initial run of the animation, so the animation plays a total of two times. The default value is "0", which means no repetition.  android:repeatMode  *int*. How an animation behaves when it reaches the end of the animation. android:repeatCount must be set to a positive integer or "-1" for this attribute to have an effect. Set to "reverse" to have the animation reverse direction with each iteration or "restart" to have the animation loop from the beginning each time.  android:valueType  *Keyword*. Do not specify this attribute if the value is a color. The animation framework automatically handles color values.   | **Value** | **Description** | | --- | --- | | intType | Specifies that the animated values are integers | | floatType (default) | Specifies that the animated values are floats |   example:  XML file saved at res/animator/property\_animator.xml:  <set android:ordering="sequentially">  <set>  <objectAnimator  android:propertyName="x"  android:duration="500"  android:valueTo="400"  android:valueType="intType"/>  <objectAnimator  android:propertyName="y"  android:duration="500"  android:valueTo="300"  android:valueType="intType"/>  </set>  <objectAnimator  android:propertyName="alpha"  android:duration="500"  android:valueTo="1f"/>  </set>  In order to run this animation, you must inflate the XML resources in your code to an [AnimatorSet](https://developer.android.com/reference/android/animation/AnimatorSet) object, and then set the target objects for all of the animations before starting the animation set. Calling [setTarget()](https://developer.android.com/reference/android/animation/AnimatorSet#setTarget(java.lang.Object)) sets a single target object for all children of the [AnimatorSet](https://developer.android.com/reference/android/animation/AnimatorSet) as a convenience. The following code shows how to do this:  AnimatorSet set = (AnimatorSet) AnimatorInflater.loadAnimator(myContext,  R.animator.property\_animator);  set.setTarget(myObject);  set.start(); |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

| **View animation** The view animation framework supports both tween and frame by frame animations, which can both be declared in XML. The following sections describe how to use both methods. **Tween animation** An animation defined in XML that performs transitions such as rotating, fading, moving, and stretching on a graphic.  file location:  res/anim/*filename*.xml  The filename will be used as the resource ID.  compiled resource datatype:  Resource pointer to an [Animation](https://developer.android.com/reference/android/view/animation/Animation).  resource reference:  In Java: R.anim.*filename*  In XML: @[*package*:]anim/*filename*  syntax:  <?xml version="1.0" encoding="utf-8"?>  <[**set**](https://developer.android.com/guide/topics/resources/animation-resource#set-element) xmlns:android="http://schemas.android.com/apk/res/android"  android:interpolator="@[package:]anim/*interpolator\_resource*"  android:shareInterpolator=["true" | "false"] >  <[**alpha**](https://developer.android.com/guide/topics/resources/animation-resource#alpha-element)  android:fromAlpha="*float*"  android:toAlpha="*float*" />  <[**scale**](https://developer.android.com/guide/topics/resources/animation-resource#scale-element)  android:fromXScale="*float*"  android:toXScale="*float*"  android:fromYScale="*float*"  android:toYScale="*float*"  android:pivotX="*float*"  android:pivotY="*float*" />  <[**translate**](https://developer.android.com/guide/topics/resources/animation-resource#translate-element)  android:fromXDelta="*float*"  android:toXDelta="*float*"  android:fromYDelta="*float*"  android:toYDelta="*float*" />  <[**rotate**](https://developer.android.com/guide/topics/resources/animation-resource#rotate-element)  android:fromDegrees="*float*"  android:toDegrees="*float*"  android:pivotX="*float*"  android:pivotY="*float*" />  <[**set**](https://developer.android.com/guide/topics/resources/animation-resource#set-element)>  ...  </set>  </set>  The file must have a single root element: either an <alpha>, <scale>, <translate>, <rotate>, or <set> element that holds a group (or groups) of other animation elements (even nested <set> elements).  elements:  <set>  A container that holds other animation elements (<alpha>, <scale>, <translate>, <rotate>) or other <set> elements. Represents an [AnimationSet](https://developer.android.com/reference/android/view/animation/AnimationSet).  attributes:  android:interpolator  *Interpolator resource*. An [Interpolator](https://developer.android.com/reference/android/view/animation/Interpolator) to apply on the animation. The value must be a reference to a resource that specifies an interpolator (not an interpolator class name). There are default interpolator resources available from the platform or you can create your own interpolator resource. See the discussion below for more about [Interpolators](https://developer.android.com/guide/topics/resources/animation-resource#Interpolators).  android:shareInterpolator  *Boolean*. "true" if you want to share the same interpolator among all child elements.  <alpha>  A fade-in or fade-out animation. Represents an [AlphaAnimation](https://developer.android.com/reference/android/view/animation/AlphaAnimation).  attributes:  android:fromAlpha  *Float*. Starting opacity offset, where 0.0 is transparent and 1.0 is opaque.  android:toAlpha  *Float*. Ending opacity offset, where 0.0 is transparent and 1.0 is opaque.  For more attributes supported by <alpha>, see the [Animation](https://developer.android.com/reference/android/view/animation/Animation) class reference (of which, all XML attributes are inherited by this element).  <scale>  A resizing animation. You can specify the center point of the image from which it grows outward (or inward) by specifying pivotX and pivotY. For example, if these values are 0, 0 (top-left corner), all growth will be down and to the right. Represents a [ScaleAnimation](https://developer.android.com/reference/android/view/animation/ScaleAnimation).  attributes:  android:fromXScale  *Float*. Starting X size offset, where 1.0 is no change.  android:toXScale  *Float*. Ending X size offset, where 1.0 is no change.  android:fromYScale  *Float*. Starting Y size offset, where 1.0 is no change.  android:toYScale  *Float*. Ending Y size offset, where 1.0 is no change.  android:pivotX  *Float*. The X coordinate to remain fixed when the object is scaled.  android:pivotY  *Float*. The Y coordinate to remain fixed when the object is scaled.  For more attributes supported by <scale>, see the [Animation](https://developer.android.com/reference/android/view/animation/Animation) class reference (of which, all XML attributes are inherited by this element).  <translate>  A vertical and/or horizontal motion. Supports the following attributes in any of the following three formats: values from -100 to 100 ending with "%", indicating a percentage relative to itself; values from -100 to 100 ending in "%p", indicating a percentage relative to its parent; a float value with no suffix, indicating an absolute value. Represents a [TranslateAnimation](https://developer.android.com/reference/android/view/animation/TranslateAnimation).  attributes:  android:fromXDelta  *Float or percentage*. Starting X offset. Expressed either: in pixels relative to the normal position (such as "5"), in percentage relative to the element width (such as "5%"), or in percentage relative to the parent width (such as "5%p").  android:toXDelta  *Float or percentage*. Ending X offset. Expressed either: in pixels relative to the normal position (such as "5"), in percentage relative to the element width (such as "5%"), or in percentage relative to the parent width (such as "5%p").  android:fromYDelta  *Float or percentage*. Starting Y offset. Expressed either: in pixels relative to the normal position (such as "5"), in percentage relative to the element height (such as "5%"), or in percentage relative to the parent height (such as "5%p").  android:toYDelta  *Float or percentage*. Ending Y offset. Expressed either: in pixels relative to the normal position (such as "5"), in percentage relative to the element height (such as "5%"), or in percentage relative to the parent height (such as "5%p").  For more attributes supported by <translate>, see the [Animation](https://developer.android.com/reference/android/view/animation/Animation) class reference (of which, all XML attributes are inherited by this element).  <rotate>  A rotation animation. Represents a [RotateAnimation](https://developer.android.com/reference/android/view/animation/RotateAnimation).  attributes:  android:fromDegrees  *Float*. Starting angular position, in degrees.  android:toDegrees  *Float*. Ending angular position, in degrees.  android:pivotX  *Float or percentage*. The X coordinate of the center of rotation. Expressed either: in pixels relative to the object's left edge (such as "5"), in percentage relative to the object's left edge (such as "5%"), or in percentage relative to the parent container's left edge (such as "5%p").  android:pivotY  *Float or percentage*. The Y coordinate of the center of rotation. Expressed either: in pixels relative to the object's top edge (such as "5"), in percentage relative to the object's top edge (such as "5%"), or in percentage relative to the parent container's top edge (such as "5%p").  For more attributes supported by <rotate>, see the [Animation](https://developer.android.com/reference/android/view/animation/Animation) class reference (of which, all XML attributes are inherited by this element).  example:  XML file saved at res/anim/hyperspace\_jump.xml:  <set xmlns:android="http://schemas.android.com/apk/res/android"  android:shareInterpolator="false">  <scale  android:interpolator="@android:anim/accelerate\_decelerate\_interpolator"  android:fromXScale="1.0"  android:toXScale="1.4"  android:fromYScale="1.0"  android:toYScale="0.6"  android:pivotX="50%"  android:pivotY="50%"  android:fillAfter="false"  android:duration="700" />  <set  android:interpolator="@android:anim/accelerate\_interpolator"  android:startOffset="700">  <scale  android:fromXScale="1.4"  android:toXScale="0.0"  android:fromYScale="0.6"  android:toYScale="0.0"  android:pivotX="50%"  android:pivotY="50%"  android:duration="400" />  <rotate  android:fromDegrees="0"  android:toDegrees="-45"  android:toYScale="0.0"  android:pivotX="50%"  android:pivotY="50%"  android:duration="400" />  </set>  </set>  This application code will apply the animation to an [ImageView](https://developer.android.com/reference/android/widget/ImageView) and start the animation:  ImageView image = (ImageView) findViewById(R.id.image);  Animation hyperspaceJump = AnimationUtils.[loadAnimation](https://developer.android.com/reference/android/view/animation/AnimationUtils#loadAnimation(android.content.Context,%20int))(this, R.anim.hyperspace\_jump);  image.[startAnimation](https://developer.android.com/reference/android/view/View#startAnimation(android.view.animation.Animation))(hyperspaceJump); |
| --- |

| **Interpolators** An interpolator is an animation modifier defined in XML that affects the rate of change in an animation. This allows your existing animation effects to be accelerated, decelerated, repeated, bounced, etc.  An interpolator is applied to an animation element with the android:interpolator attribute, the value of which is a reference to an interpolator resource.  All interpolators available in Android are subclasses of the [Interpolator](https://developer.android.com/reference/android/view/animation/Interpolator) class. For each interpolator class, Android includes a public resource you can reference in order to apply the interpolator to an animation using the android:interpolator attribute. The following table specifies the resource to use for each interpolator:   | **Interpolator class** | **Resource ID** | | --- | --- | | [AccelerateDecelerateInterpolator](https://developer.android.com/reference/android/view/animation/AccelerateDecelerateInterpolator) | @android:anim/accelerate\_decelerate\_interpolator | | [AccelerateInterpolator](https://developer.android.com/reference/android/view/animation/AccelerateInterpolator) | @android:anim/accelerate\_interpolator | | [AnticipateInterpolator](https://developer.android.com/reference/android/view/animation/AnticipateInterpolator) | @android:anim/anticipate\_interpolator | | [AnticipateOvershootInterpolator](https://developer.android.com/reference/android/view/animation/AnticipateOvershootInterpolator) | @android:anim/anticipate\_overshoot\_interpolator | | [BounceInterpolator](https://developer.android.com/reference/android/view/animation/BounceInterpolator) | @android:anim/bounce\_interpolator | | [CycleInterpolator](https://developer.android.com/reference/android/view/animation/CycleInterpolator) | @android:anim/cycle\_interpolator | | [DecelerateInterpolator](https://developer.android.com/reference/android/view/animation/DecelerateInterpolator) | @android:anim/decelerate\_interpolator | | [LinearInterpolator](https://developer.android.com/reference/android/view/animation/LinearInterpolator) | @android:anim/linear\_interpolator | | [OvershootInterpolator](https://developer.android.com/reference/android/view/animation/OvershootInterpolator) | @android:anim/overshoot\_interpolator |   Here's how you can apply one of these with the android:interpolator attribute:  <set android:interpolator="@android:anim/accelerate\_interpolator">  ...  </set> **Custom interpolators** If you're not satisfied with the interpolators provided by the platform (listed in the table above), you can create a custom interpolator resource with modified attributes. For example, you can adjust the rate of acceleration for the [AnticipateInterpolator](https://developer.android.com/reference/android/view/animation/AnticipateInterpolator), or adjust the number of cycles for the [CycleInterpolator](https://developer.android.com/reference/android/view/animation/CycleInterpolator). In order to do so, you need to create your own interpolator resource in an XML file.  file location:  res/anim/*filename*.xml  The filename will be used as the resource ID.  compiled resource datatype:  Resource pointer to the corresponding interpolator object.  resource reference:  In XML: @[*package*:]anim/*filename*  syntax:  <?xml version="1.0" encoding="utf-8"?>  <*InterpolatorName* xmlns:android="http://schemas.android.com/apk/res/android"  android:*attribute\_name*="*value*"  />  If you don't apply any attributes, then your interpolator will function exactly the same as those provided by the platform (listed in the table above).  elements:  Notice that each [Interpolator](https://developer.android.com/reference/android/view/animation/Interpolator) implementation, when defined in XML, begins its name in lowercase.  <accelerateDecelerateInterpolator>  The rate of change starts and ends slowly but accelerates through the middle.  No attributes.  <accelerateInterpolator>  The rate of change starts out slowly, then accelerates.  attributes:  android:factor  *Float*. The acceleration rate (default is 1).  <anticipateInterpolator>  The change starts backward then flings forward.  attributes:  android:tension  *Float*. The amount of tension to apply (default is 2).  <anticipateOvershootInterpolator>  The change starts backward, flings forward and overshoots the target value, then settles at the final value.  attributes:  android:tension  *Float*. The amount of tension to apply (default is 2).  android:extraTension  *Float*. The amount by which to multiply the tension (default is 1.5).  <bounceInterpolator>  The change bounces at the end.  No attributes  <cycleInterpolator>  Repeats the animation for a specified number of cycles. The rate of change follows a sinusoidal pattern.  attributes:  android:cycles  *Integer*. The number of cycles (default is 1).  <decelerateInterpolator>  The rate of change starts out quickly, then decelerates.  attributes:  android:factor  *Float*. The deceleration rate (default is 1).  <linearInterpolator>  The rate of change is constant.  No attributes.  <overshootInterpolator>  The change flings forward and overshoots the last value, then comes back.  attributes:  android:tension  *Float*. The amount of tension to apply (default is 2).  example:  XML file saved at res/anim/my\_overshoot\_interpolator.xml:  <?xml version="1.0" encoding="utf-8"?>  <overshootInterpolator xmlns:android="http://schemas.android.com/apk/res/android"  android:tension="7.0"  />  This animation XML will apply the interpolator:  <scale xmlns:android="http://schemas.android.com/apk/res/android"  android:interpolator="@anim/my\_overshoot\_interpolator"  android:fromXScale="1.0"  android:toXScale="3.0"  android:fromYScale="1.0"  android:toYScale="3.0"  android:pivotX="50%"  android:pivotY="50%"  android:duration="700" /> |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

| **Frame animation** An animation defined in XML that shows a sequence of images in order (like a film).  file location:  res/drawable/*filename*.xml  The filename will be used as the resource ID.  compiled resource datatype:  Resource pointer to an [AnimationDrawable](https://developer.android.com/reference/android/graphics/drawable/AnimationDrawable).  resource reference:  In Java: R.drawable.*filename*  In XML: @[*package*:]drawable.*filename*  syntax:  <?xml version="1.0" encoding="utf-8"?>  <[**animation-list**](https://developer.android.com/guide/topics/resources/animation-resource#animation-list-element) xmlns:android="http://schemas.android.com/apk/res/android"  android:oneshot=["true" | "false"] >  <[**item**](https://developer.android.com/guide/topics/resources/animation-resource#item-element)  android:drawable="@[package:]drawable/*drawable\_resource\_name*"  android:duration="*integer*" />  </animation-list>  elements:  <animation-list>  **Required**. This must be the root element. Contains one or more <item> elements.  attributes:  android:oneshot  *Boolean*. "true" if you want to perform the animation once; "false" to loop the animation.  <item>  A single frame of animation. Must be a child of a <animation-list> element.  attributes:  android:drawable  *Drawable resource*. The drawable to use for this frame.  android:duration  *Integer*. The duration to show this frame, in milliseconds.  example:  XML file saved at res/drawable/rocket\_thrust.xml:  <?xml version="1.0" encoding="utf-8"?>  <animation-list xmlns:android="http://schemas.android.com/apk/res/android"  android:oneshot="false">  <item android:drawable="@drawable/rocket\_thrust1" android:duration="200" />  <item android:drawable="@drawable/rocket\_thrust2" android:duration="200" />  <item android:drawable="@drawable/rocket\_thrust3" android:duration="200" />  </animation-list>  This application code will set the animation as the background for a View, then play the animation:  [Kotlin](https://developer.android.com/guide/topics/resources/animation-resource#kotlin)  [Java](https://developer.android.com/guide/topics/resources/animation-resource#java)  ImageView rocketImage = (ImageView) findViewById(R.id.rocket\_image);  rocketImage.[**setBackgroundResource**](https://developer.android.com/reference/android/view/View#setBackgroundResource(int))(R.drawable.rocket\_thrust);  rocketAnimation = rocketImage.[**getBackground()**](https://developer.android.com/reference/android/view/View#getBackground());  if (rocketAnimation instanceof [**Animatable**](https://developer.android.com/reference/android/graphics/drawable/Animatable)) {  ((Animatable)rocketAnimation).[**start()**](https://developer.android.com/reference/android/graphics/drawable/Animatable#start()); |
| --- |