Layout Resource

See also

- Layouts

A layout resource defines the architecture for the UI in an Activity or a component of a UI.

FILE LOCATION:

`res/layout/filename.xml`
The filename will be used as the resource ID.

COMPILED RESOURCE DATATYPE:

Resource pointer to a `View` (or subclass) resource.

RESOURCE REFERENCE:

In Java: `R.layout.filename`
In XML: `@[package:]layout/filename`

SYNTAX:

```
<?xml version="1.0" encoding="utf-8"?>
<ViewGroup
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@[+][package:]id/resource_name"
    android:layout_height=["dimension" | "match_parent" | "wrap_content"]
    android:layout_width=["dimension" | "match_parent" | "wrap_content"]
    [ViewGroup-specific attributes] >
    <View
        android:id="@[+][package:]id/resource_name"
        android:layout_height=["dimension" | "match_parent" | "wrap_content"]
        android:layout_width=["dimension" | "match_parent" | "wrap_content"]
        [View-specific attributes] >
        <requestFocus/>
    </View>
    <ViewGroup >
        <View />
    </ViewGroup>
    <include layout="@layout/layout_resource"/>
</ViewGroup>
```

Note: The root element can be either a `ViewGroup`, a `View`, or a `<merge>` element, but there must be only one root element and it must contain the `xmlns:android` attribute with the `android` namespace as shown.

ELEMENTS:

`<ViewGroup>`

A container for other `View` elements. There are many different kinds of `ViewGroup` objects and each one lets you specify the layout of the child elements in different ways. Different kinds of `ViewGroup` objects include `LinearLayout`, `RelativeLayout`, and `FrameLayout`.

You should not assume that any derivation of `ViewGroup` will accept nested `View`s. Some `ViewGroup`s are implementations of the `AdapterView` class, which determines its children only from an `Adapter`.

attributes:

`android:id`

Resource ID. A unique resource name for the element, which you can use to obtain a reference to the `ViewGroup` from your application. See more about the value for `android:id` below.

`android:layout_height`

Dimension or keyword. **Required**. The height for the group, as a dimension value (or dimension resource) or a keyword (`"match_parent"` or `"wrap_content"`). See the valid values below.

`android:layout_width`

Dimension or keyword. **Required**. The width for the group, as a dimension value (or dimension resource) or a keyword (`"match_parent"` or `"wrap_content"`). See the valid values below.

More attributes are supported by the `ViewGroup` base class, and many more are supported by each implementation of `ViewGroup`. For a reference of all available attributes, see the corresponding reference documentation for the `ViewGroup` class (for example, the LinearLayout XML attributes).

`<View>`

An individual UI component, generally referred to as a "widget". Different kinds of `View` objects include `TextView`, `Button`, and `CheckBox`.

attributes:

`android:id`

Resource ID. A unique resource name for the element, which you can use to obtain a reference to the `View` from your application. See more about the value for `android:id` below.

`android:layout_height`

Dimension or keyword. **Required**. The height for the element, as a dimension value (or dimension resource) or a keyword (`"match_parent"` or `"wrap_content"`). See the valid values below.

`android:layout_width`

Dimension or keyword. **Required**. The width for the element, as a dimension value (or dimension resource) or a keyword (`"match_parent"` or `"wrap_content"`). See the valid values below.

More attributes are supported by the `View` base class, and many more are supported by each implementation of `View`. Read Layouts for more information. For a reference of all available attributes, see the corresponding reference documentation (for example, the TextView XML attributes).

`<requestFocus>`

Any element representing a `View` object can include this empty element, which gives its parent initial focus on the screen. You can have only one of these elements per file.

`<include>`

Includes a layout file into this layout.

attributes:

`layout`

Layout resource. **Required**. Reference to a layout resource.

`android:id`

Resource ID. Overrides the ID given to the root view in the included layout.

`android:layout_height`

Dimension or keyword. Overrides the height given to the root view in the included layout. Only effective if `android:layout_width` is also declared.

`android:layout_width`

Dimension or keyword. Overrides the width given to the root view in the included layout. Only effective if `android:layout_height` is also declared.

You can include any other layout attributes in the `<include>` that are supported by the root element in the included layout and they will override those defined in the root element.

> **Caution:** If you want to override layout attributes using the `<include>` tag, you must override both `android:layout_height` and `android:layout_width` in order for other layout attributes to take effect.

Another way to include a layout is to use `ViewStub`. It is a lightweight View that consumes no layout space until you explicitly inflate it, at which point, it includes a layout file defined by its `android:layout` attribute. For more information about using `ViewStub`, read Loading Views On Demand.

`<merge>`

An alternative root element that is not drawn in the layout hierarchy. Using this as the root element is useful when you know that this layout will be placed into a layout that already contains the appropriate parent View to contain the children of the `<merge>` element. This is particularly useful when you plan to include this layout in another layout file using `<include>` and this layout doesn't require a different `ViewGroup` container. For more information about merging layouts, read Re-using Layouts with <include/>.

Value for `android:id`

For the ID value, you should usually use this syntax form: `"@+id/name"`. The plus symbol, `+`, indicates that this is a new resource ID and the `aapt` tool will create a new resource integer in the `R.java` class, if it doesn't already exist. For example:

```
<TextView android:id="@+id/nameTextbox"/>
```

The `nameTextbox` name is now a resource ID attached to this element. You can then refer to the `TextView` to which the ID is associated in Java:

```
findViewById(R.id.nameTextbox);
```

This code returns the `TextView` object.

However, if you have already defined an ID resource (and it is not already used), then you can apply that ID to a `View` element by excluding the plus symbol in the `android:id` value.

Value for `android:layout_height` and `android:layout_width`:

The height and width value can be expressed using any of the dimension units supported by Android (px, dp, sp, pt, in, mm) or with the following keywords:

Value	Description

`match_parent`	Sets the dimension to match that of the parent element. Added in API Level 8 to deprecate `fill_parent`.
`wrap_content`	Sets the dimension only to the size required to fit the content of this element.

Custom View elements

You can create your own custom `View` and `ViewGroup` elements and apply them to your layout the same as a standard layout element. You can also specify the attributes supported in the XML element. To learn more, see the Custom Components developer guide.

EXAMPLE:

XML file saved at `res/layout/main_activity.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="match_parent"
              android:layout_height="match_parent"
              android:orientation="vertical" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello, I am a Button" />
</LinearLayout>
```

This application code will load the layout for an `Activity`, in the `onCreate()` method:

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

SEE ALSO:

- Layouts
- `View`
- `ViewGroup`