More Resource Types

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

Bool

XML resource that carries a boolean value.

Color

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

Dimension

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

ID

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

Integer

XML resource that carries an integer value.

Integer Array

XML resource that provides an array of integers.

Typed Array

XML resource that provides a `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>
    <bool
        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();
boolean screenIsSmall = res.getBoolean(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>
    <color
        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();
int color = res.getColor(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; each 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>
    <dimen
        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();
float fontSize = res.getDimension(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` 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>
    <item
        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(R.id.dialog_exit);
```

In the same application, the "dialog_exit" ID is compared when creating a dialog:

```
protected Dialog 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>
    <integer
        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();
int maxSpeed = res.getInteger(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>
    <integer-array
        name="integer_array_name">
        <item
            >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();
int[] bits = res.getIntArray(R.array.bits);
```

Typed Array

A `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`'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`.

RESOURCE REFERENCE:

In Java: `R.array.`*`array_name`*
In XML: `@[`*`package`*`:]array.`*`array_name`*

SYNTAX:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <array
        name="integer_array_name">
        <item>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();
TypedArray icons = res.obtainTypedArray(R.array.icons);
Drawable drawable = icons.getDrawable(0);

TypedArray colors = res.obtainTypedArray(R.array.colors);
int color = colors.getColor(0,0);
```