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 (Plurals)
: XML resource that carries different strings for different quantities of the same word or phrase.

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` will be 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"?>
<resources>
    <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 will be 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)` or `getText(int)` to retieve a string. `getText(int)` will retain 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` will be used as the resource ID.

COMPILED RESOURCE DATATYPE:

Resource pointer to an array of `String`s.

RESOURCE REFERENCE:

In Java: `R.array.string_array_name`

SYNTAX:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array
        name="string_array_name">
        <item
            >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 will be used as the resource ID to reference the array.

`<item>`
A string, which can include styling tags. The value can be a referenced 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 Styling, below, for information about to 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 string array:

```
Resources res = getResources();
String[] planets = res.getStringArray(R.array.planets_array);
```

Quantity Strings (Plurals)

Different languages have different rules for grammatical agreement with quantity. In English, for example, the quantity 1 is a special case. We write "1 book", but for any other quantity we'd write "*n* books". This distinction between singular and plural is very common, but other languages make finer distinctions. The full set supported by Android is `zero`, `one`, `two`, `few`, `many`, and `other`.

The rules for deciding which case to use for a given language and quantity can be very complex, so Android provides you with methods such as `getQuantityString()` to select the appropriate resource for you.

Note that the selection is made based on grammatical necessity. A string for `zero` in English will be ignored even if the quantity is 0, because 0 isn't grammatically different from 2, or any other number except 1 ("zero books", "one book", "two books", et cetera). Don't be misled either by the fact that, say, `two` sounds like it could only apply to the quantity 2: a language may require that 2, 12, 102 (et cetera) are all treated like one another but differently to other quantities. Rely on your translator to know what distinctions their language actually insists upon.

It's often possible to avoid quantity strings by using quantity-neutral formulations such as "Books: 1". This will make your life and your translators' lives easier, if it's a style that's in keeping with your application.

> **Note:** A plurals collection 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 plurals 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 `<plurals>` element's `name` will be used as the resource ID.

RESOURCE REFERENCE:

In Java: `R.plurals.plural_name`

SYNTAX:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <plurals
        name="plural_name">
        <item
            quantity=["zero" | "one" | "two" | "few" | "many" | "other"]
            >text_string</item>
    </plurals>
</resources>
```

ELEMENTS:

`<resources>`
Required. This must be the root node.

No attributes.

`<plurals>`
A collection of strings, of which, one string is provided depending on the amount of something. Contains one or more `<item>` elements.

ATTRIBUTES:

`name`
String. A name for the pair of strings. This name will be used as the resource ID.

`<item>`
A plural or singular string. The value can be a referenced to another string resource. Must be a child of a `<plurals>` element. Beware that you must escape apostrophes and quotation marks. See Formatting and Styling, below, for information about to properly style and format your strings.

ATTRIBUTES:

`quantity`
Keyword. A value indicating when this string should be used. Valid values, with non-exhaustive examples in parentheses:

Value	Description
`zero`	When the language requires special treatment of the number 0 (as in Arabic).
`one`	When the language requires special treatment of numbers like one (as with the number 1 in English and most other languages; in Russian, any number ending in 1 but not ending in 11 is in this class).
`two`	When the language requires special treatment of numbers like two (as in Welsh).
`few`	When the language requires special treatment of "small" numbers (as with 2, 3, and 4 in Czech; or numbers ending 2, 3, or 4 but not 12, 13, or 14 in Polish).
`many`	When the language requires special treatment of "large" numbers (as with numbers ending 11-99 in Maltese).
`other`	When the language does not require special treatment of the given quantity.

EXAMPLE:

XML file saved at `res/values/strings.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <plurals name="numberOfSongsAvailable">
        <item quantity="one">One song found.</item>
        <item quantity="other">%d songs found.</item>
    </plurals>
</resources>
```

XML file saved at `res/values-pl/strings.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <plurals name="numberOfSongsAvailable">
        <item quantity="one">Znaleziono jedną piosenkę.</item>
        <item quantity="few">Znaleziono %d piosenki.</item>
        <item quantity="other">Znaleziono %d piosenek.</item>
    </plurals>
</resources>
```

Java code:

```
int count = getNumberOfsongsAvailable();
Resources res = getResources();
String songsFound = res.getQuantityString(R.plurals.numberOfSongsAvailable,
count);
```

Formatting and Styling

Here are a few important things you should know about how to properly format and style your string resources.

Escaping apostrophes and quotes

If you have an apostrophe or a quote in your string, you must either escape it or enclose the whole string in the other type of enclosing quotes. For example, here are some stings that do and don't work:

```
<string name="good_example">"This'll work"</string>
<string name="good_example_2">This\'ll also work</string>
<string name="bad_example">This doesn't work</string>
<string name="bad_example_2">XML encodings don&apos;t work</string>
```

Formatting strings

If you need to format your strings using `String.format(String, Object...)`, then you can do so by putting your format arguments in the string resource. For example, with the following resource:

```
<string name="welcome_messages">Hello, %1$s! You have %2$d new messages.</string>
```

In this example, the format string has two arguments: `%1$s` is a string and `%2$d` is a decimal number. You can format the string with arguements from your application like this:

```
Resources res = getResources();
String text = String.format(res.getString(R.string.welcome_messages), username,
mailCount);
```

Styling with HTML markup

You can add styling to your strings with HTML markup. For example:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="welcome">Welcome to <b>Android</b>!</string>
</resources>
```

Supported HTML elements include:

- <b> for **bold** text.
- <i> for *italic* text.
- <u> for underline text.

Sometimes you may want to create a styled text resource that is also used as a format string. Normally, this won't work because the String.format(String, Object...) method will strip all the style information from the string. The work-around to this is to write the HTML tags with escaped entities, which are then recovered with fromHtml(String), after the formatting takes place. For example:

1. Store your styled text resource as an HTML-escaped string:

   ```
   <resources>
     <string name="welcome_messages">Hello, %1$s! You have &lt;b>%2$d new
   messages&lt;/b>.</string>
   </resources>
   ```

   In this formatted string, a <b> element is added. Notice that the opening bracket is HTML-escaped, using the < notation.
2. Then format the string as usual, but also call fromHtml(String) to convert the HTML text into styled text:

   ```
   Resources res = getResources();
   String text = String.format(res.getString(R.string.welcome_messages), username,
   mailCount);
   CharSequence styledText = Html.fromHtml(text);
   ```

Because the fromHtml(String) method will format all HTML entities, be sure to escape any possible HTML characters in the strings you use with the formatted text, using htmlEncode(String). For instance, if you'll be passing a string argument to String.format() that may contain characters such as "<" or "&", then they must be escaped before formatting, so that when the formatted string is passed through fromHtml(String), the characters come out the way they were originally written. For example:

```
String escapedUsername = TextUtil.htmlEncode(username);

Resources res = getResources();
String text = String.format(res.getString(R.string.welcome_messages),
escapedUsername, mailCount);
CharSequence styledText = Html.fromHtml(text);
```

← Back to Resource Types ↑ Go to top

Except as noted, this content is licensed under Apache 2.0. For details and restrictions, see the Content License.
Android 3.1 r1 - 17 Jun 2011 10:58
Site Terms of Service - Privacy Policy - Brand Guidelines