Android compiled resources - resources.arsc

Asked 10 years ago Modified 3 years, 1 month ago Viewed 61k times

Part of Mobile Development Collective

43

I am trying to figure out what it mean to "compile resources".

What I did in order to understand this issue:

I have read many articles about the subject but didn't find a simple answer. The best one I have read was this: How does the mapping between android resources and resources ID work?.

How I understand it:

From my understanding, when we compile our project either by ANT (Eclipse) or Gradle (AS). We use a tool called **aapt** - Android Asset Packaging Tool which: Is used to generate unique IDs for each of our resources, such as our layouts, our styles and more and store them in a lookup table. Then it persists this lookup table by generating two files:

1. It Generates the R.java file with these unique IDs so we will be able to use our resources from our java code during compilation.
2. It generate the resources.arsc file which can be found in resources*.ap_ file. This resources.arsc file will later be packed by the apktool to the apk.
   This arsc file format is a format that will be easily mapped and parsed by the device at runtime.

An Example:

So to make it simple: lets say I have this in my activity_main.xml:

```
    <TextView android:id="@+id/my_textView"
        android:text="@string/hello_world"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
```

And I call it from my onCreate using:

```
findViewById(R.id.my_textView)
```

In my R.java file I will see:

```
public static final int my_textView=0x7f08003f;
```

Using: aapt dump resources on the generated apk I can see it contains two lines with my_textView: ec resource 0x7f08003f com.example.lizi.liortest2:id/my_textView: flags=0x00000000 resource 0x7f08003f com.example.lizi.liortest2:id/my_textView: t=0x12 d=0x00000000 (s=0x0008 r=0x00)

What I don't understand:

I would have thought that this resources.arsc file will not just contain the resource ID but also all the properties I have defined for the view, such as android:layout_width="wrap_content".

So now during runtime when the VM tries to run `findViewById(R.id.my_textView)` How does it know which view to get / its properties to create?

I simply can't understand how it works... Shouldn't this lookup table contain also the properties data? And what is this 0x7f08003f number? (Should it represent a value that will later be mapped to physical memory in which the object will be stored?)

MD android resources r.java-file

Share
Improve this question
Follow

edited Aug 8, 2018 at 2:00
user12205
2,702 ● 1 ● 21 ● 40

asked Dec 18, 2014 at 14:27
ZiviMagic
1,054 ● 2 ● 10 ● 25

3 Answers

Sorted by: Highest score (default)

53

***TL;DR**: With the help of android asset packagin tool(aapt), xml nodes get translated to Java classes and the corresponding xml attributes get translated to numerical Ids. Android run-time works with these numeric ids to instantiate classes and create the views*

TL;R

Run this command to dump the binary xml

`aapt d xmltree apk_file_name res/layout/activity_main.xml` (*aapt can be found in **android-sdk-dir/build-tools/23.0.2/aapt.exe***)

This will show the xml nodes (e.g. `LinearLayout`, `RelativeLayout`, etc) with their attributes(e.g. `android:layout_width, android:layout_height`) and their values. Note that, the constants `match_parent`(numeric value `0xffffffff` or `-1`) or `wrap_content`(numeric value `0xfffffffe` or `-2`) can be seen there.

As a matter of fact, you can use this command on any other xml files in the apk e.g. `AndroidManifest.xml or other layout files`

The apk file is just a zip archive containing all the java class files(`classes.dex`), all the compiled resource files and a file named `resources.arsc`. This `resource.arsc` file contains all the meta-information about the resources. Some of those are...

- the xml nodes(e.g. `LinearLayout`, `RelativeLayout`, etc),
- the attributes(e.g. `android:layout_width`),
- the resource `id`'s.

The resource `id`'s refer to the real resources in the apk-file. The attributes are resolved to a value at runtime. The resolution process is smart about any re-direction (`@dimen/...` as opposed to `4dp` or `@color/...` as opposed to `"#FFaabbcc"`) and returns a usable value(a `dimen` value is resolved differently than a `color` value).

Whats a compiled XML file: A compiled XML file is just the same XML file with the resource references changed to their corresponding `ids`. For example, a reference `@string/ok` will be replaced by `0x7f000001`. Moreover, the attributes from `android` namespace is changed to their respective integer values(e.g. `wrap_content` is changed to `0xfffffffe` or `-2`)

How Android resolves resources at runtime: The method `inflater.inflate()` parses a compiled xml file and creates a view hierarchy by instantiating the xml nodes. Each of the xml nodes is instantiated by a java class(e.g. LinearLayout.java, RelativeLayout.java). To instantiate, the inflater parses the compiled xml file, collects all the attributes of a node and creates a packed structure of type `AttributeSet`. This `AttributeSet` is passed to the class constructor. The class constructor has the responsibility of walking the `AttributeSet` and resolving each of the attribute values.

For example, for a layout containing `RelativeLayout`, the `inflater` will pack `layout_width` and `layout_height` into a `AttributeSet` and pass it to the constructor

RelativeLayout(Context context, AttributeSet attrs, int defStyleAttr, int defStyleRes).

In this case, some of the attributes are resolved by RelativeLayout.initFromAttributes(). The rest of the attributes are resolved by the parent ViewGroup.initFromAttributes(). The attribute `android:id` of a view is just

another attribute. After inflating, the inflater stores the id of each view by calling `setId(id)` on that view after instantiation

Now to answer your question `R.id` is a java array and `R.id.my_textview` is an integer in that array. The `id` of the view `my_textview` is this integer(starts with 0x7f). The method `findViewById()` does a depth-first search on that view hierarchy to find the respective view.

Hope this helps. The link you provided in your question already answers how the ids are generated by aapt.

Its a wonderful system of managing resources for devices with multiple dimensions of variations. Moreover, the implementation is really fast !! With this as the foundation, it allows to implement higher level functionality(e.g. Runtime Resource Overlay)

Share

Improve this answer

Follow

edited Oct 24, 2021 at 18:12

answered Jun 10, 2015 at 22:22

pellucide

3,617 ● 2 ● 23 ● 25

9 What a great answer. Are you an Android? – Clive Jefferies Aug 1, 2017 at 15:21

Thanks! How to extract strings? `strings.xml` doesn't exist there (as well as `values` folder). I know, a searched string exists in binary `resources.arsc` file, but I cannot extract it from there. Can crackers extract any resources (strings, arrays, colors) from there?
– CoolMind Jul 10, 2020 at 9:41

2

LayoutInflater inflate view by using XML strings. XML strings compiled into resource file as you mentioned in your question.

Please check these code snippets of AOSP:

```
public View inflate(int resource, ViewGroup root, boolean attachToRoot) {
    final Resources res = getContext().getResources();
    if (DEBUG) {
        Log.d(TAG, "INFLATING from resource: \"" +
res.getResourceName(resource) + "\" ("
                + Integer.toHexString(resource) + ")");
    }

    final XmlResourceParser parser = res.getLayout(resource);
    try {
        return inflate(parser, root, attachToRoot);
    } finally {
        parser.close();
    }
}
```

`Resources.getLayout` loads XML resource parser

```
public XmlResourceParser getLayout(int id) throws NotFoundException {
    return loadXmlResourceParser(id, "layout");
}

XmlResourceParser loadXmlResourceParser(int id, String type)
        throws NotFoundException {
    synchronized (mAccessLock) {
        TypedValue value = mTmpValue;
        if (value == null) {
            mTmpValue = value = new TypedValue();
        }
        getValue(id, value, true);
        if (value.type == TypedValue.TYPE_STRING) {
            return loadXmlResourceParser(value.string.toString(), id,
                    value.assetCookie, type);
        }
        throw new NotFoundException(
                "Resource ID #0x" + Integer.toHexString(id) + " type #0x"
                + Integer.toHexString(value.type) + " is not valid");
    }
}
```

`getValue` uses AssetManager's getResourceValue and it calls `loadResourceValue` native method. This native method calls `ResTable`'s getResource method to get XML strings stored in resource file.

Share Improve this answer Follow

answered Jun 13, 2015 at 4:31

Wonil

6,717 ● 2 ● 43 ● 57

1

```
Use appt for android-sdk (ex:- /build-tools/27.0.3/aapt )

 run given script and get resources.arsc file content
./aapt dump resources ./debug.apk

Package Groups (1)
Package Group 0 id=0x7f packageCount=1 name=com.dianping.example.activity
  Package 0 id=0x7f name=com.dianping.example.activity
    type 1 configCount=3 entryCount=1
      spec resource 0x7f020000 com.example.activity:drawable/ic_launcher:
flags=0x00000100
      config mdpi-v4:
        resource 0x7f020000 com.example.activity:drawable/ic_launcher: t=0x03
d=0x00000000 (s=0x0008 r=0x00)
      config hdpi-v4:
        resource 0x7f020000 com.example.activity:drawable/ic_launcher: t=0x03
d=0x00000001 (s=0x0008 r=0x00)
      config xhdpi-v4:
        resource 0x7f020000 com.example.activity:drawable/ic_launcher: t=0x03
d=0x00000002 (s=0x0008 r=0x00)
    type 2 configCount=1 entryCount=1
      spec resource 0x7f030000 com.dianping.example.activity:string/app_name:
flags=0x00000000
      config (default):
```

```
        resource 0x7f030000 com.dianping.example.activity:string/app_name:
t=0x03 d=0x00000003 (s=0x0008 r=0x00)
```

This link might help http://elinux.org/Android_aapt

Share Improve this answer Follow

answered Sep 9, 2018 at 17:46

Tamilan C.Periyasamy

343 ● 3 ● 8