Shrinking your APK, By Example

A Little About Me

Games Team @ NYTimes

NYT Crossword on the Play Store

@benlikestocode

Why Shrink Your APK?

Why Care About APK Size?

People pay for mobile data, and effectively are **paying to download your app**

If you're building an **Instant App** you have a hard 4MB limit per feature module

Decreasing your app size **can increase your conversion rate** when trying to get new users to download[1]

[1] https://goo.gl/zuGrX6

Strategies

- Remove Unused Resources
- Convert Resources to Smaller Formats
- Use Proguard and the Resource Shrinker
- Tune Your Proguard & Shrinking
- Use Density/ABI Splits

Use the APK Analyzer as you go along

Our Example App

A Simple Example App

An app for cat pictures with two flavors, free and paid

One activity is used **in both flavors**

Another activity is a **paid cat picture activity** 😻🤑

Initially, we're **not using any shrinking or other tools**

A Simple Example App

An app for cat pictures with two flavors, free and paid

One activity is used **in both flavors**

Another activity is a **paid cat picture activity** 😻🤑

Initially, we're **not using any shrinking or other tools**

Follow Along on Github

https://goo.gl/S5DQWP

The APK Analyzer

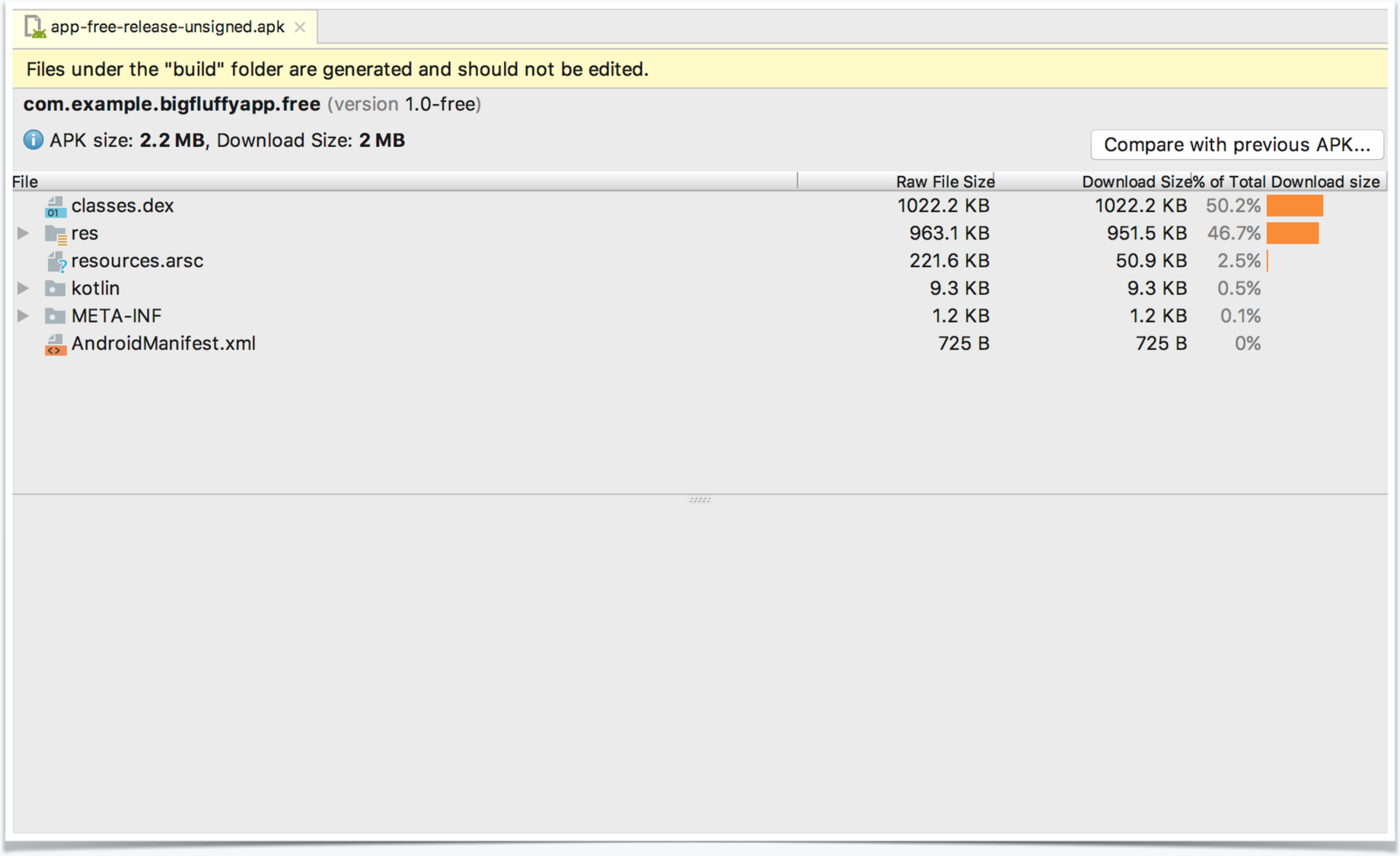
APK Analyzer is a GUI tool in Android Studio, as well as a command line utility

It offers you some useful **size metrics** for your APK

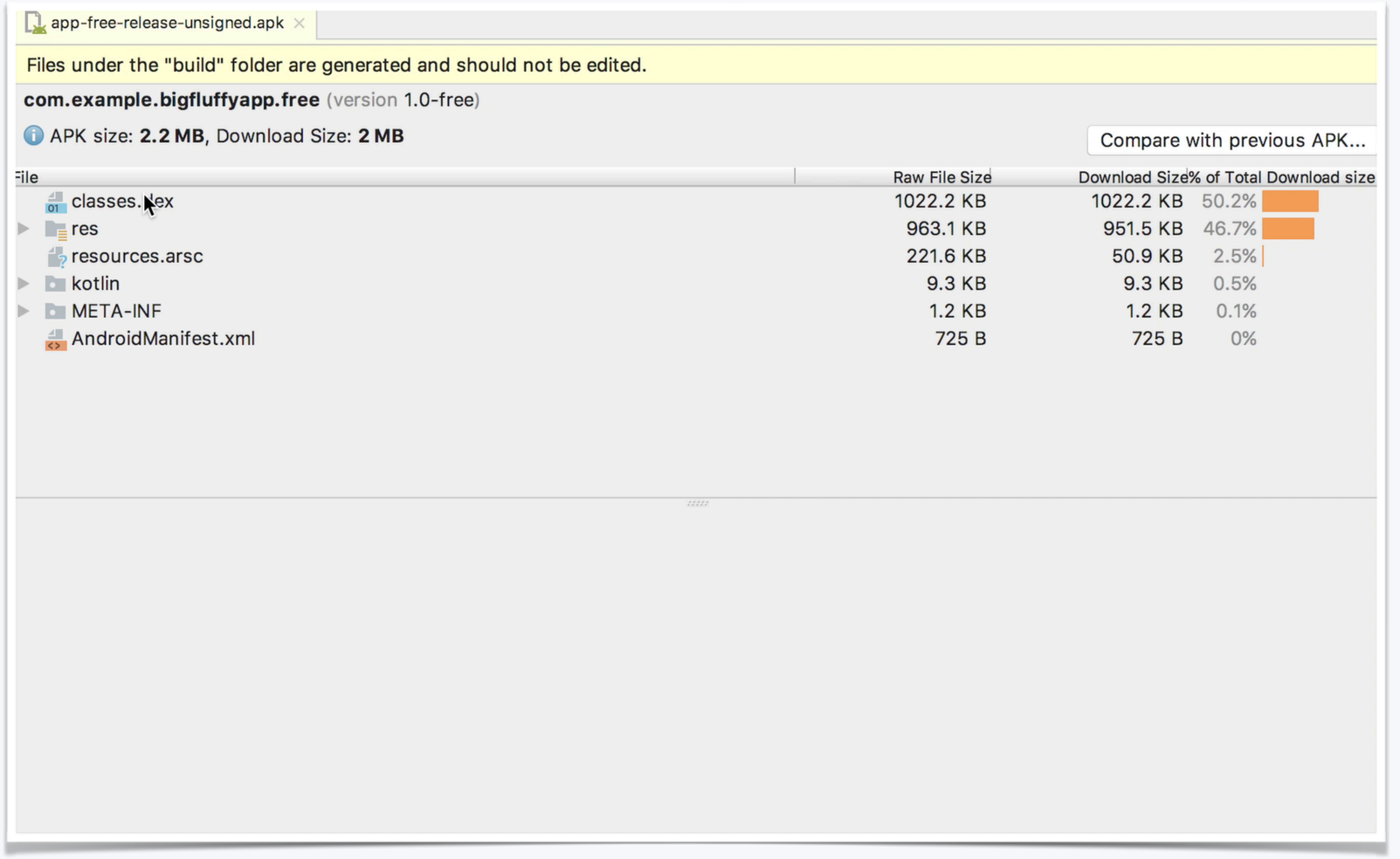
You can also inspect **resources within the APK**

And also inspect **the dex bytecode in the APK**

A Peek With The APK Analyzer

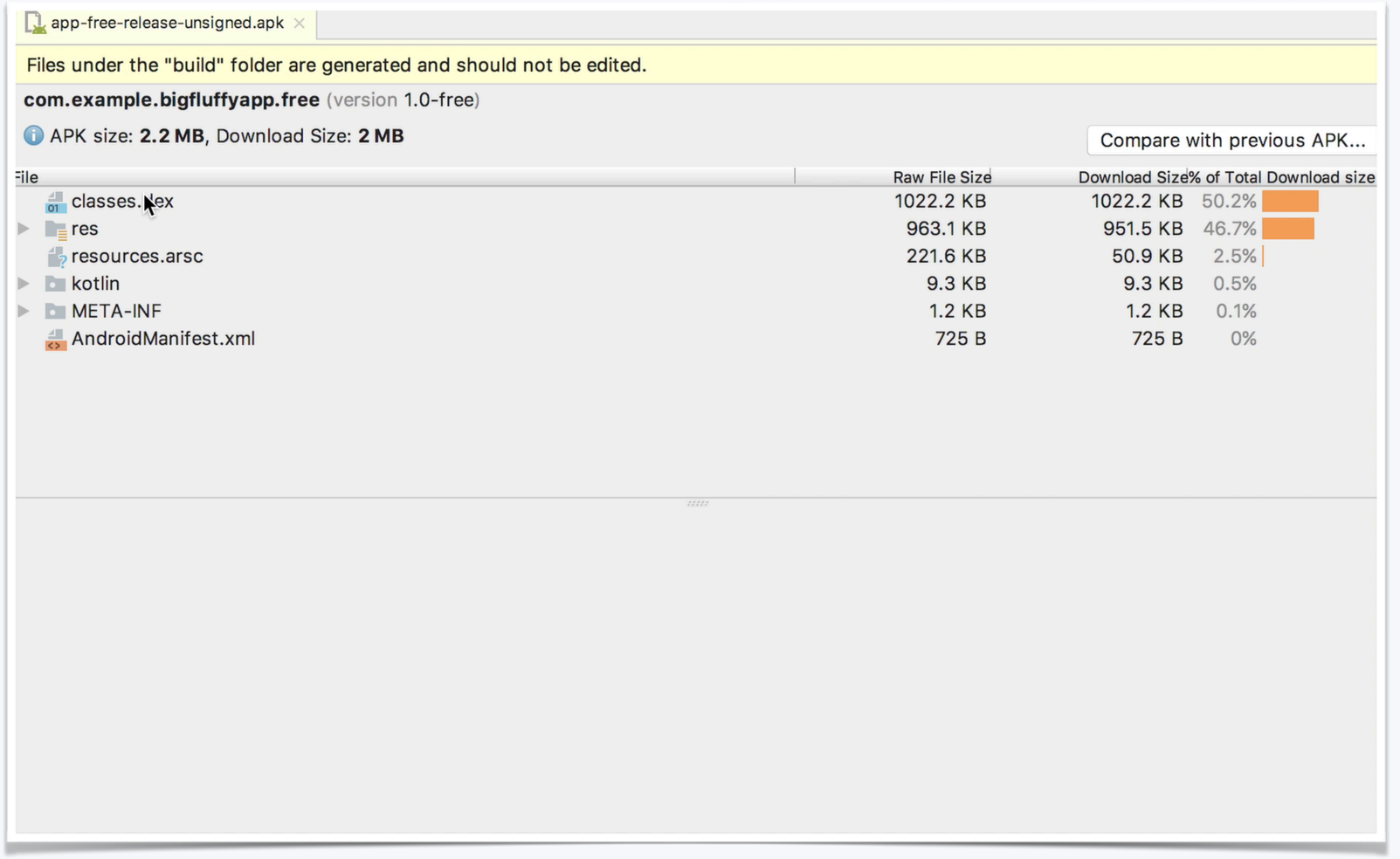


A Peek With The APK Analyzer



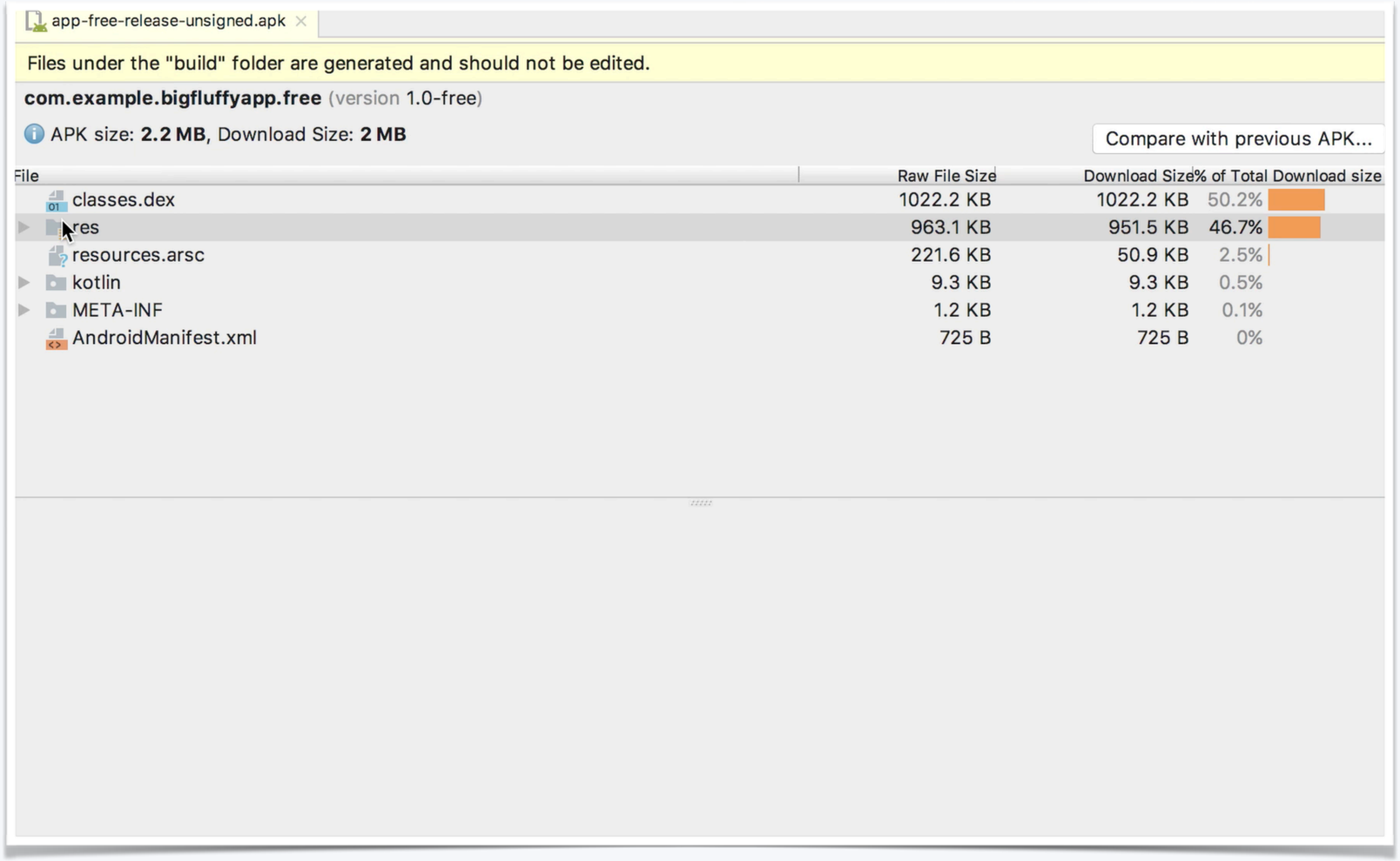
Explore the code in your dex file

A Peek With The APK Analyzer



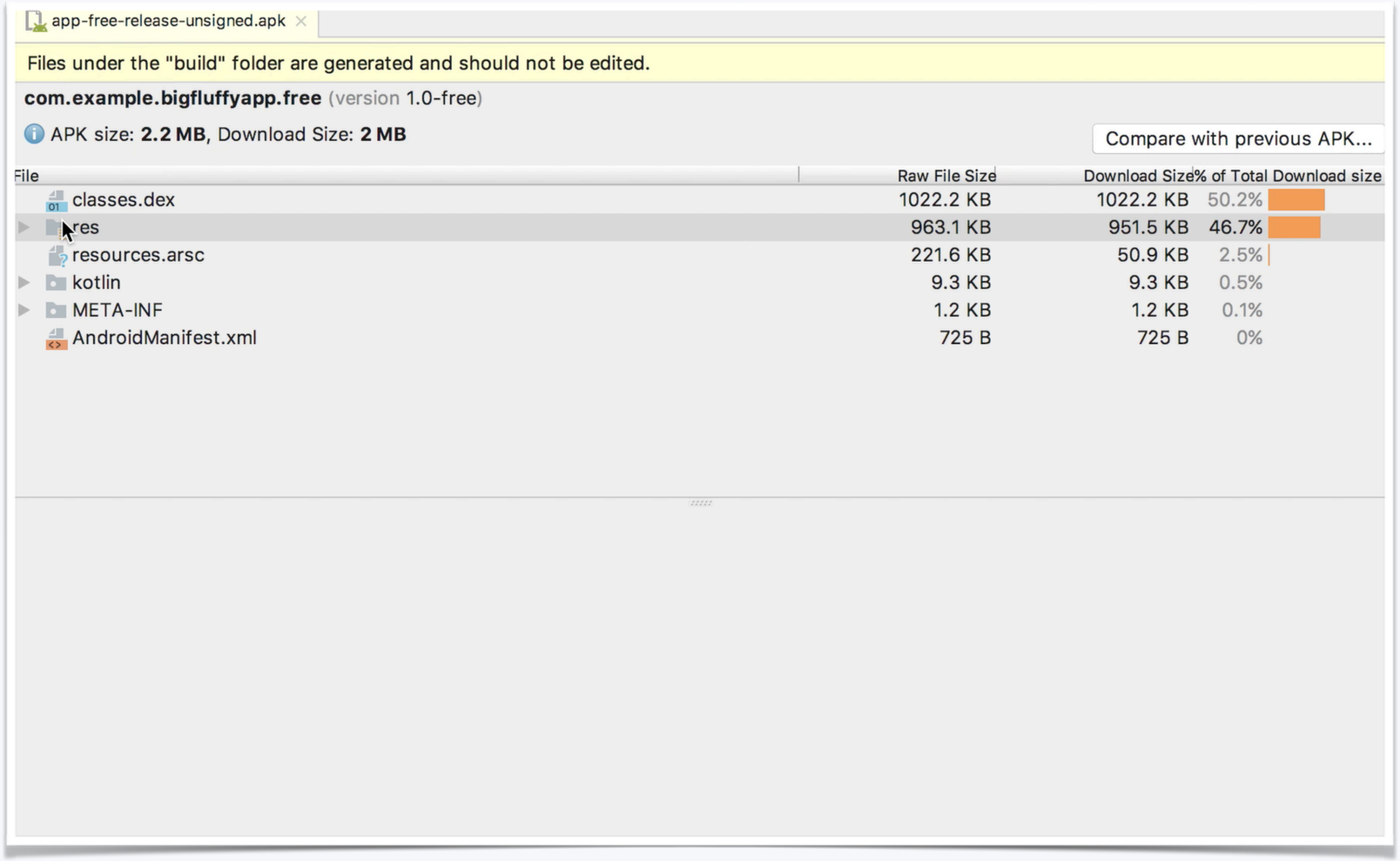
Explore the code in your dex file

A Peek With The APK Analyzer



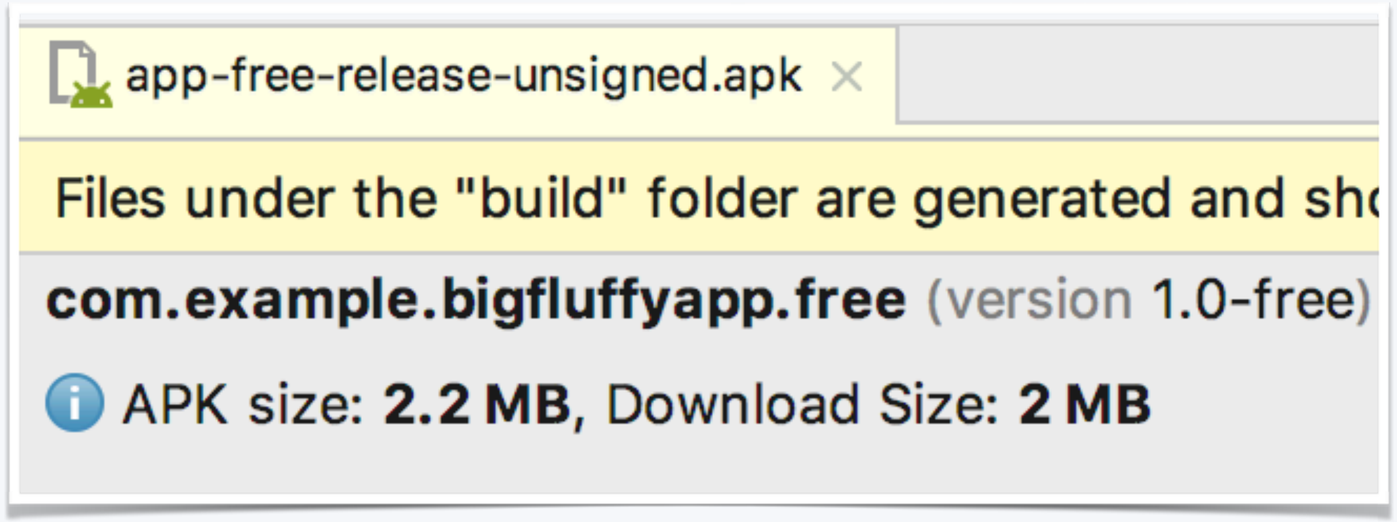
Explore the resources in your APK

A Peek With The APK Analyzer



Explore the resources in your APK

Our Baseline



Let’s beat 2.2 MB

Removing Altogether Unused Resources

Altogether Unused Resources

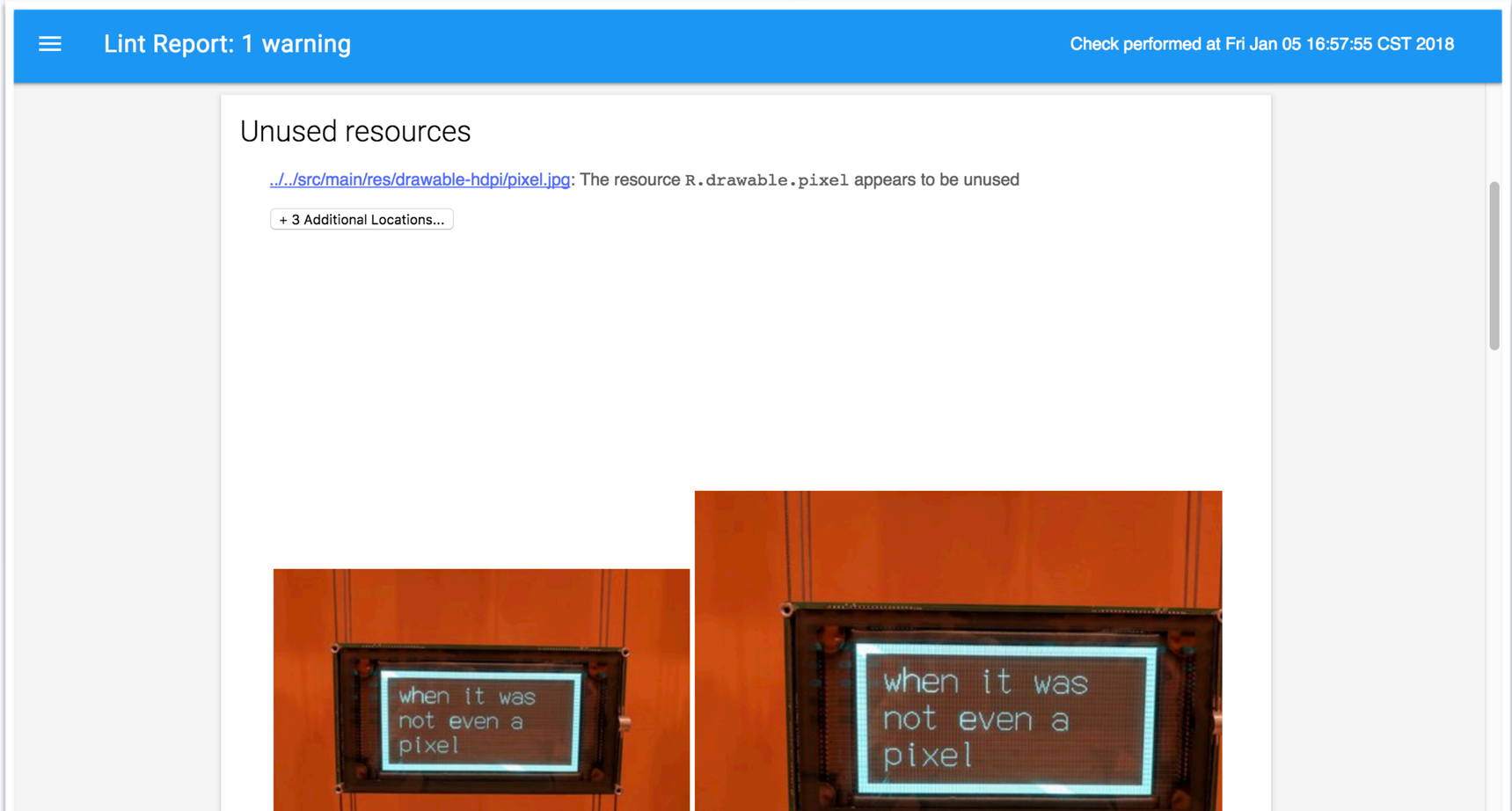
Maybe got that asset from a designer, put it in the project, and **changed your mind**

Maybe you deleted code in the past, and **forgot to remove the corresponding assets**

We've all been there, **that's why Android provides lint checks to resolve this**

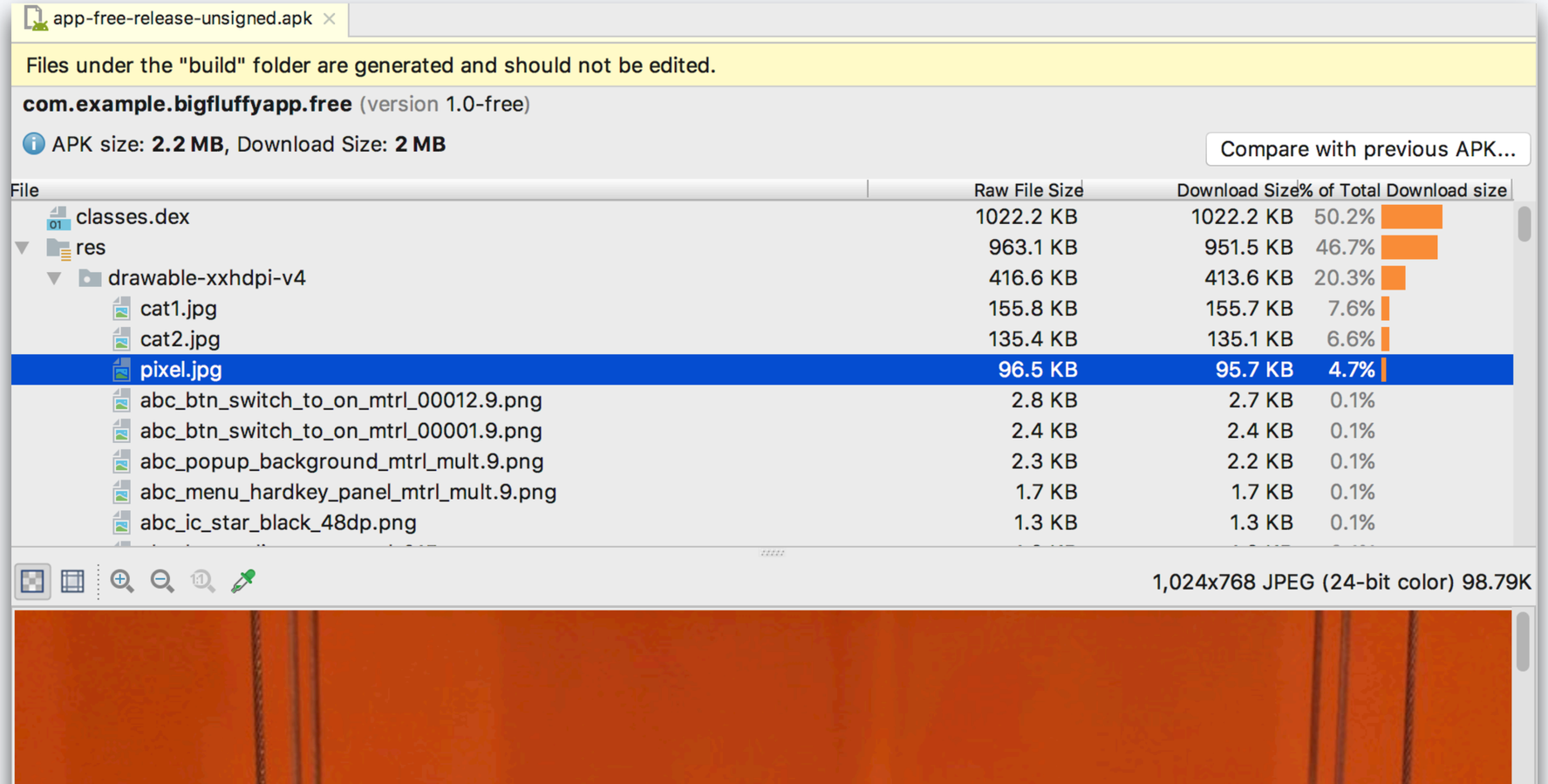
Use Your Lint Checks!

Android Lint has inspections for unused resources.



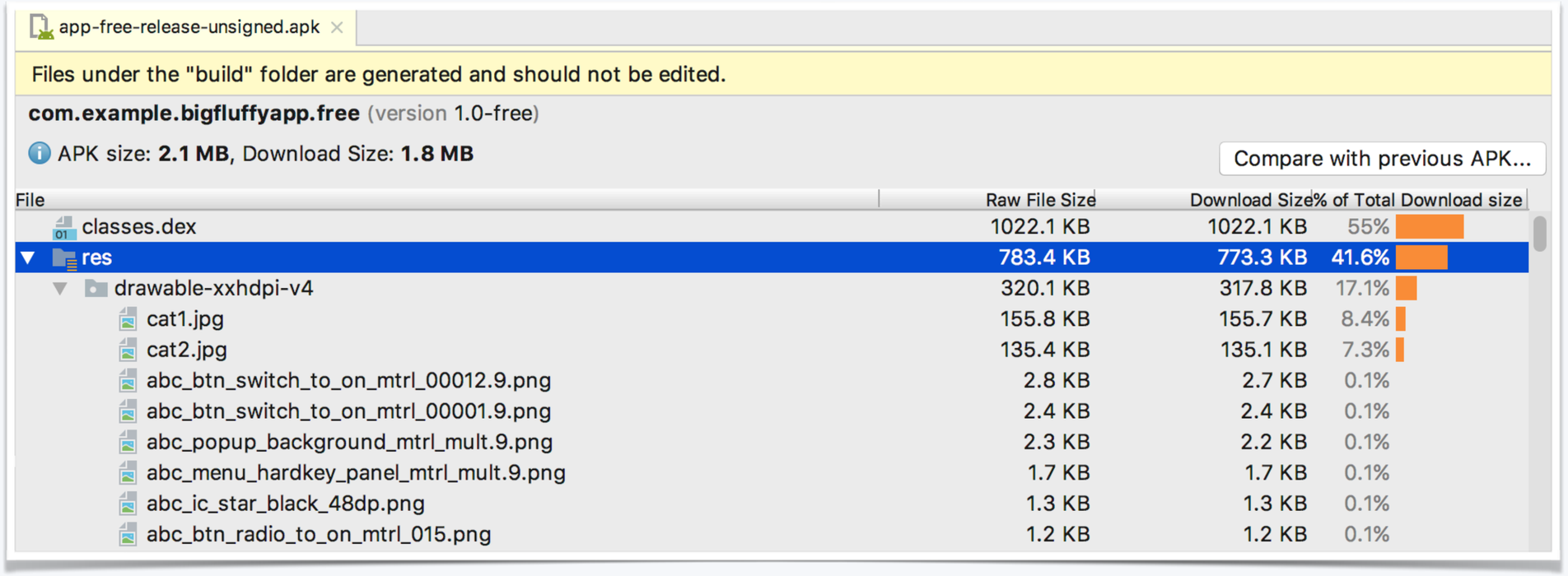
Use Your Lint Checks!

Here's that image resource taking up space in our APK.



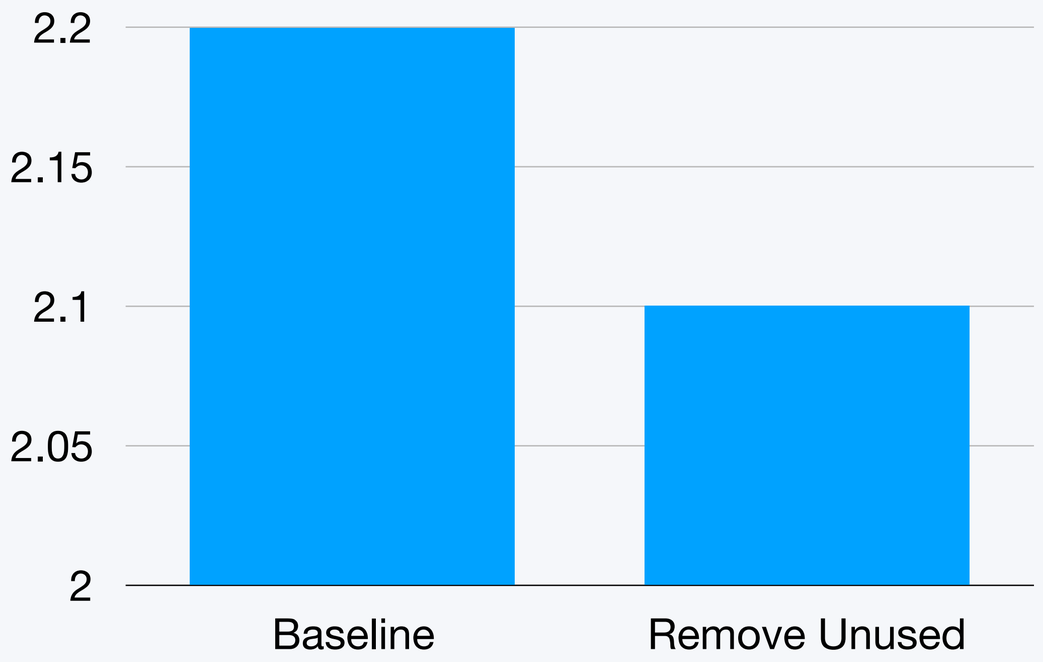
Use Your Lint Checks!

Remove that image **and we saved ~100KB vs baseline.**



Use Your Lint Checks!

Remove that image **and we saved ~100KB vs baseline.**



Converting Resources to Smaller Formats

Convert Resources to Smaller Formats

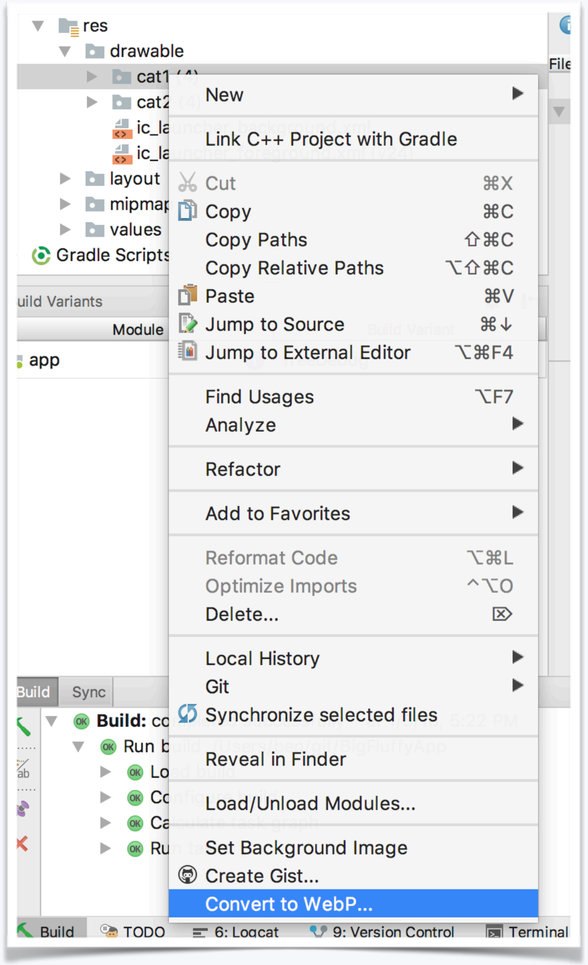
Raster images can be **converted to WebP**

Those vector images you had otherwise converted to raster, **can be VectorDrawables**

Consider audio assets as well; **Ogg and AAC sound better at lower bitrates**

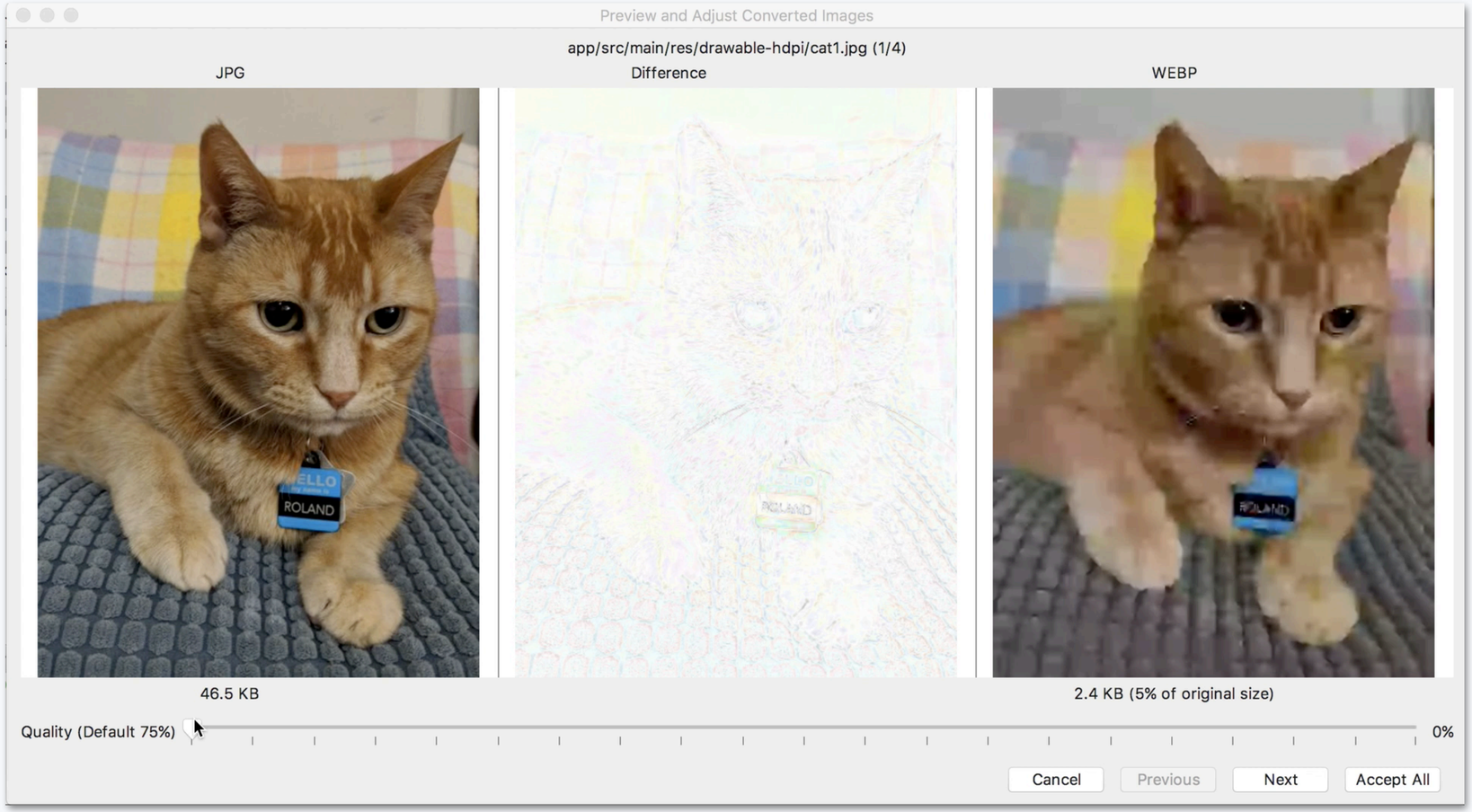
Converting to WebP

Studio has a quick tool to **convert image assets to WebP**



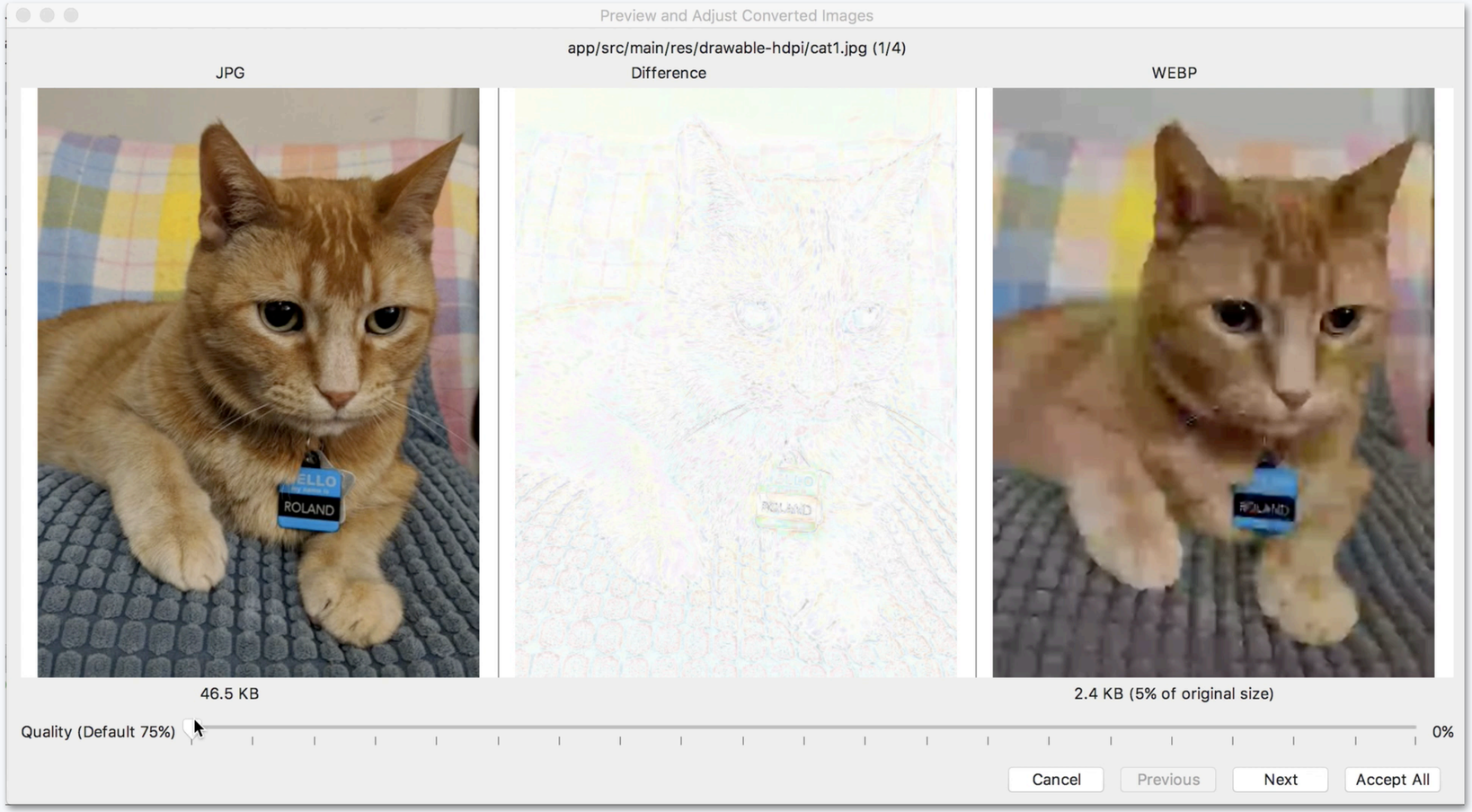
Converting to WebP

Studio has a quick tool to **convert image assets to WebP**



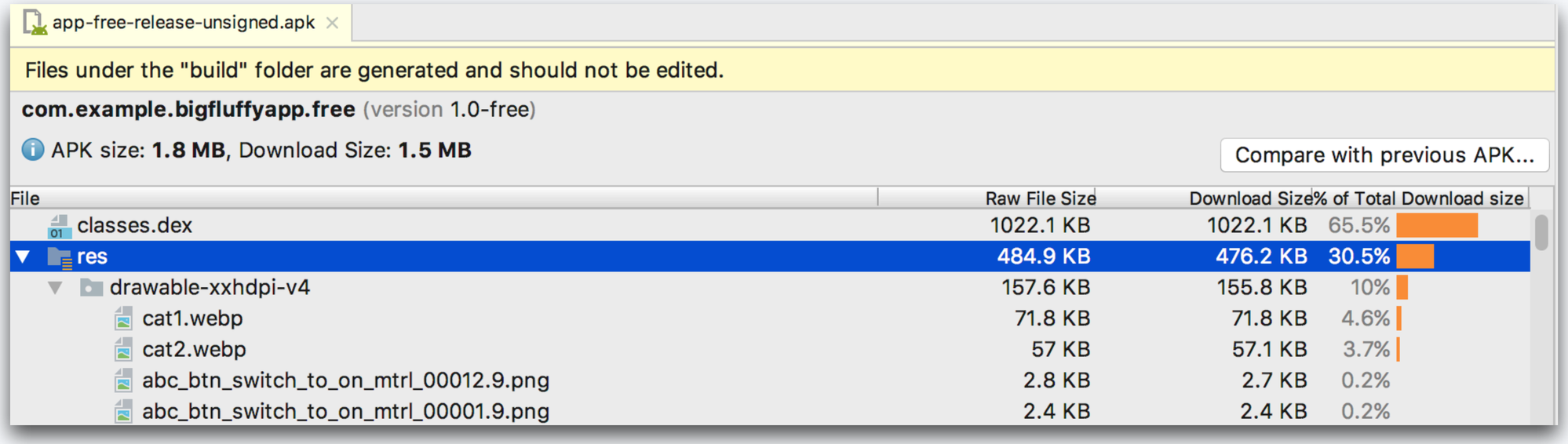
Converting to WebP

Studio has a quick tool to **convert image assets to WebP**



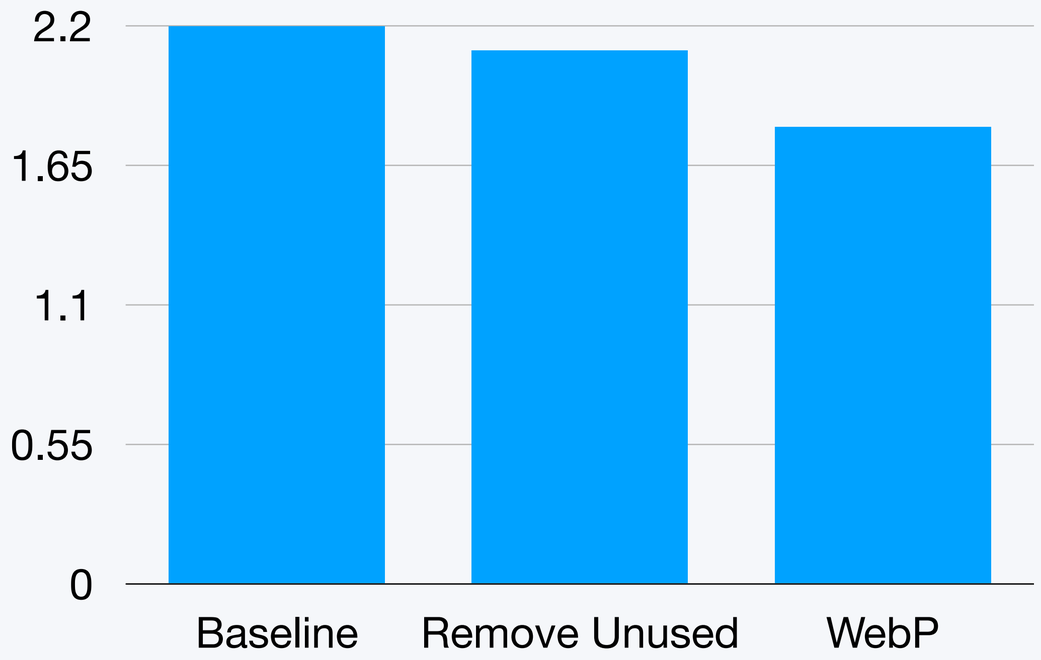
Converting to WebP

Convert those image assets to WebP at 75% compression
and we've cumulatively saved 400KB relative to baseline.



Converting to WebP

Convert those image assets to WebP at 75% compression and we've cumulatively saved **400KB** relative to baseline.



Enabling ProGuard & Resouce Shrinking

Enable ProGuard

You'll notice we focused solely on **resources right now**

There was still **1MB worth of dex in our little app**

As well as **unused resources brought in by the support library**

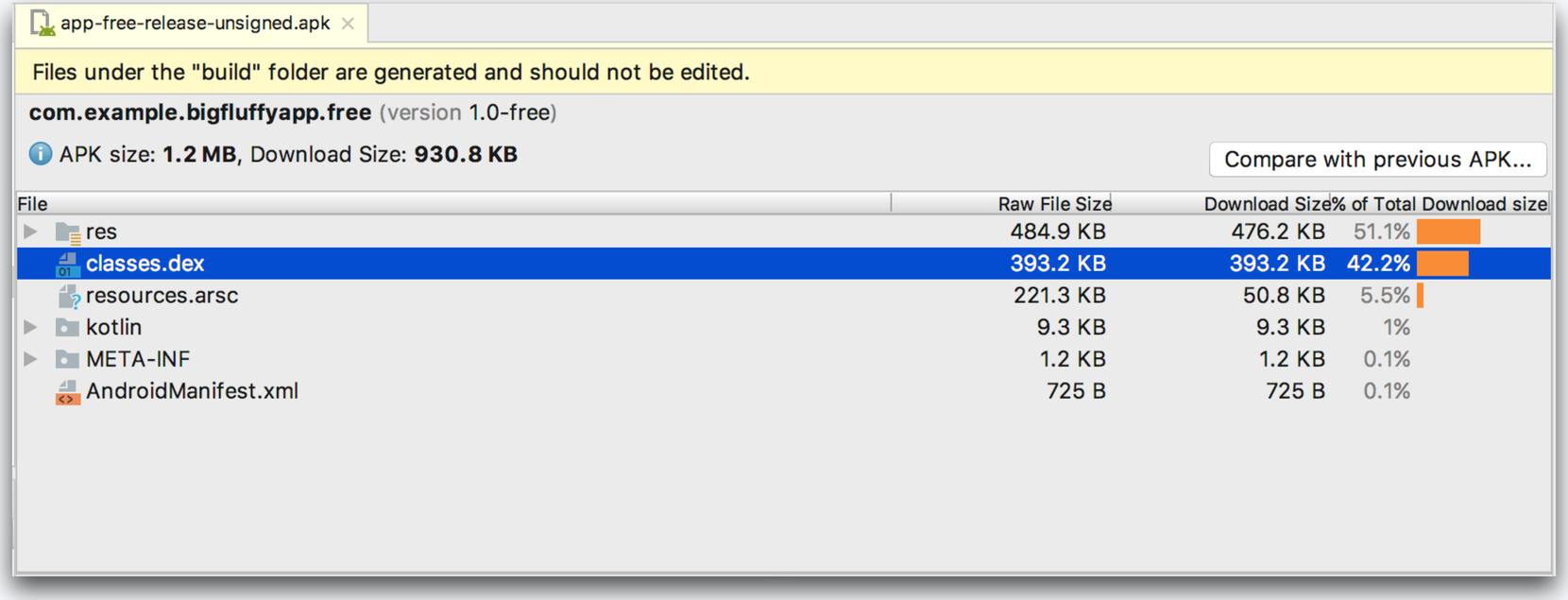
This is a totally contrived toy example, but real apps have unused library code too!

Enable ProGuard

```
buildTypes {
    release {
        minifyEnabled true
        proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-
rules.pro'
    }
}
```

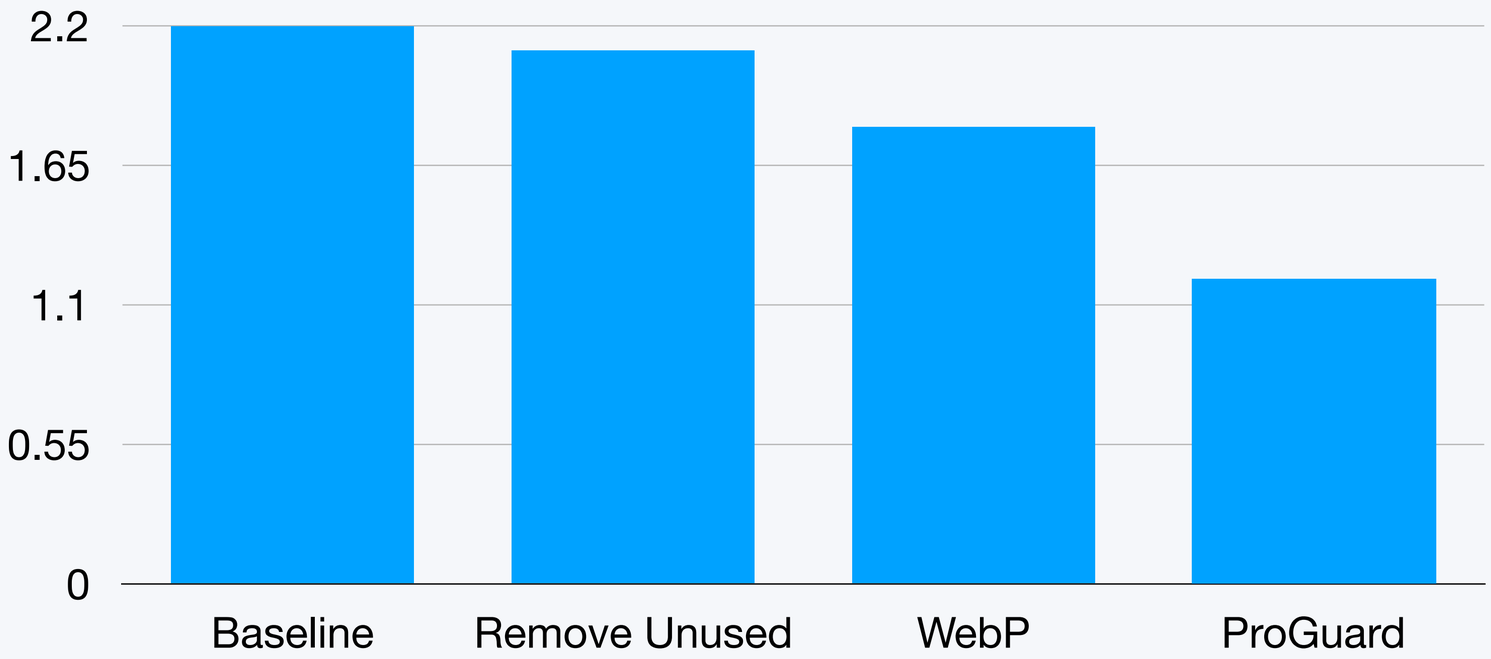
Enabling ProGuard

With ProGuard we stripped unused code, and cumulatively we've saved 1MB relative to baseline.



Enabling ProGuard

With ProGuard we stripped unused code, and cumulatively we've saved 1MB relative to baseline.



Enable the Resource Shrinker

The toolchain also has code to **shrink resources in your app**

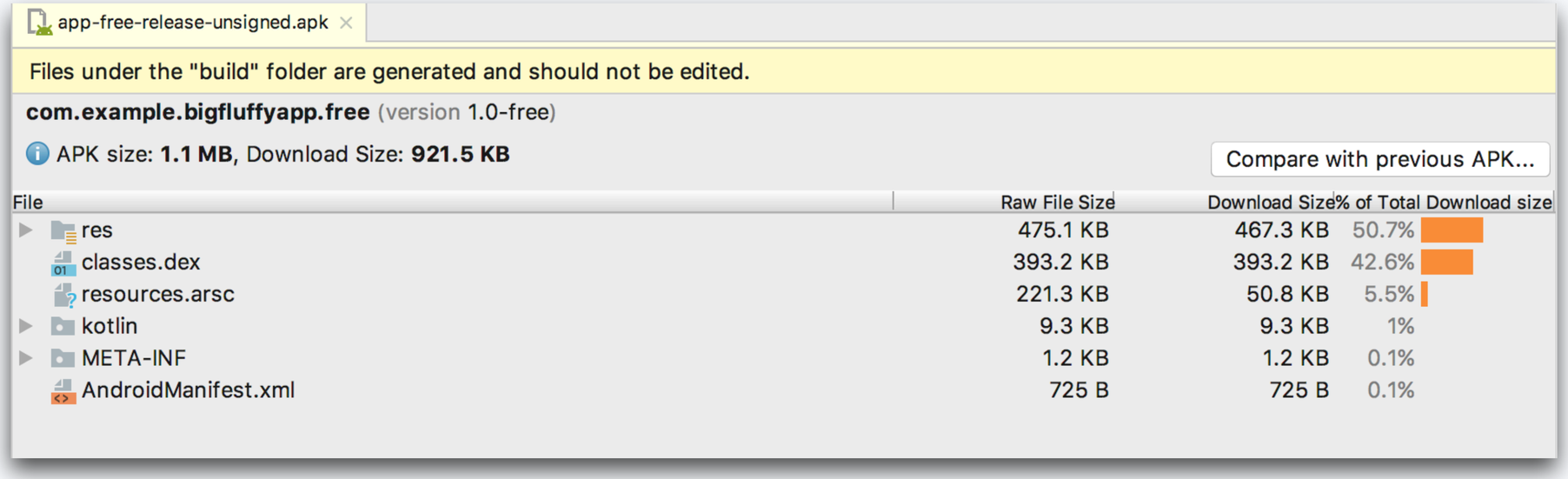
It searches for references to resources in your code and layouts, **and removes unreferenced resources**

It will **strip the resources** referenced by unused library code that ProGuard removed

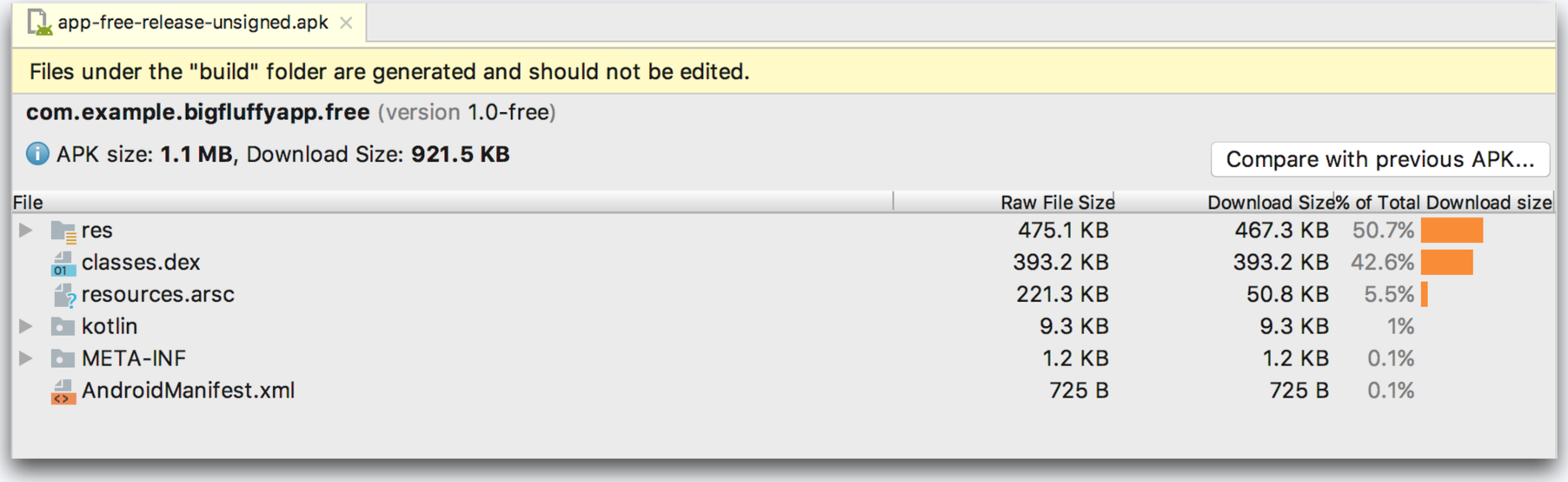
Enable the Resource Shrinker

```
buildTypes {
    release {
        minifyEnabled true
        shrinkResources true
        proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-
rules.pro'
    }
}
```

Enable the Resource Shrinker



Enable the Resource Shrinker



Wait a sec, **that didn't really help too much?** **10KB savings** 😢

Optimizing ProGuard & Shrinking

MainActivity

```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val freeActivityButton : Button = findViewById(R.id.click_for_cat_picture)
        val paidActivityButton : Button = findViewById(R.id.click_for_another_cat_picture)

        if (FlavorConfig.ALLOW_PAID_PICTURES) {
            paidActivityButton.visibility = VISIBLE

            paidActivityButton.setOnClickListener {
                launchPaidCatPicture()
            }

        } else {
            paidActivityButton.visibility = GONE
        }

        freeActivityButton.setOnClickListener {
            launchFreeCatPicture()
        }
    }

    fun launchFreeCatPicture() {
        val intent = Intent(this, FreePictureActivity::class.java)
        startActivity(intent)
    }

    fun launchPaidCatPicture() {
        val intent = Intent(this, PaidPictureActivity::class.java)
        startActivity(intent)
    }

}
```

MainActivity

False

```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val freeActivityButton : Button = findViewById(R.id.click_for_cat_picture)
        val paidActivityButton : Button = findViewById(R.id.click_for_another_cat_picture)

        if (FlavorConfig.ALLOW_PAID_PICTURES) {
            paidActivityButton.visibility = VISIBLE

            paidActivityButton.setOnClickListener {
                launchPaidCatPicture()
            }

        } else {
            paidActivityButton.visibility = GONE
        }

        freeActivityButton.setOnClickListener {
            launchFreeCatPicture()
        }
    }

    fun launchFreeCatPicture() {
        val intent = Intent(this, FreePictureActivity::class.java)
        startActivity(intent)
    }

    fun launchPaidCatPicture() {
        val intent = Intent(this, PaidPictureActivity::class.java)
        startActivity(intent)
    }

}
```

MainActivity

```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val freeActivityButton : Button = findViewById(R.id.click_for_cat_picture)
        val paidActivityButton : Button = findViewById(R.id.click_for_another_cat_picture)

        if (FlavorConfig.ALLOW_PAID_PICTURES) {
            paidActivityButton.visibility = VISIBLE

            paidActivityButton.setOnClickListener {
                launchPaidCatPicture()
            }

        } else {
            paidActivityButton.visibility = GONE
        }

        freeActivityButton.setOnClickListener {
            launchFreeCatPicture()
        }
    }

    fun launchFreeCatPicture() {
        val intent = Intent(this, FreePictureActivity::class.java)
        startActivity(intent)
    }

    fun launchPaidCatPicture() {
        val intent = Intent(this, PaidPictureActivity::class.java)
        startActivity(intent)
    }

}
```

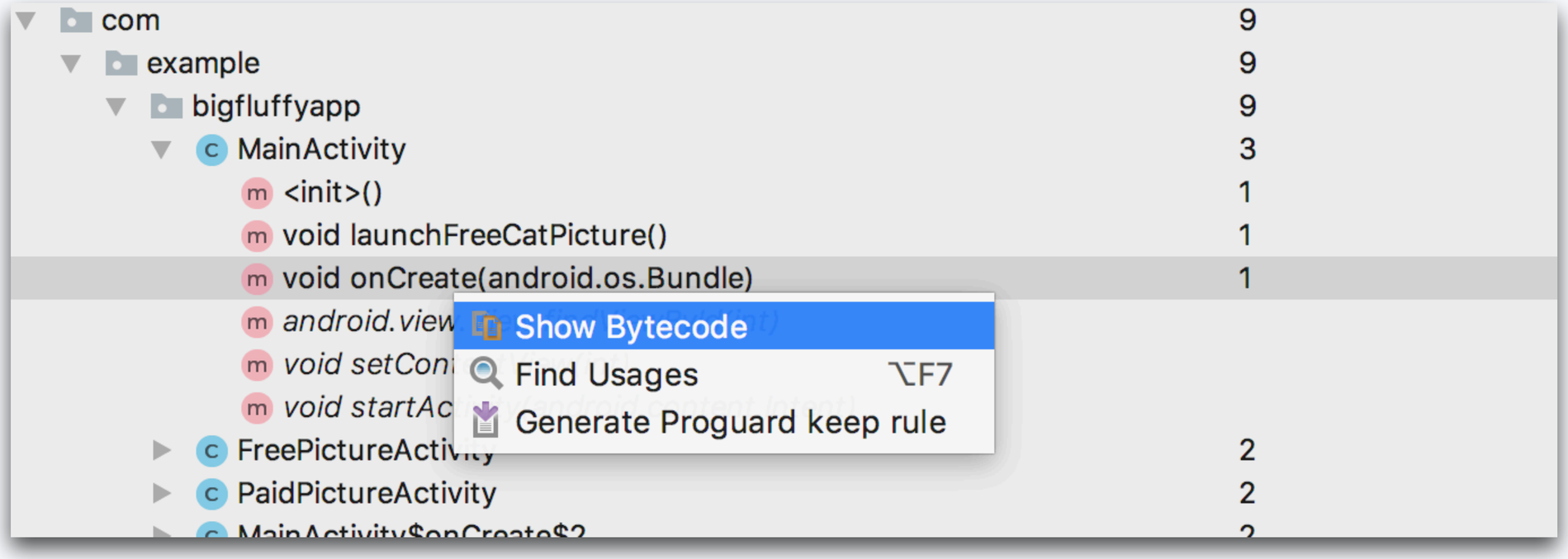
MainActivity

```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val freeActivityButton : Button = findViewById(R.id.click_for_cat_picture)
        val paidActivityButton : Button = findViewById(R.id.click_for_another_cat_picture)

        paidActivityButton.visibility = GONE

        freeActivityButton.setOnClickListener {
            launchFreeCatPicture()
        }
    }

    fun launchFreeCatPicture() {
        val intent = Intent(this, FreePictureActivity::class.java)
        startActivity(intent)
    }

    fun launchPaidCatPicture() {
        val intent = Intent(this, PaidPictureActivity::class.java)
        startActivity(intent)
    }

}
```

MainActivity

```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val freeActivityButton : Button = findViewById(R.id.click_for_cat_picture)
        val paidActivityButton : Button = findViewById(R.id.click_for_another_cat_picture)

        paidActivityButton.visibility = GONE

        freeActivityButton.setOnClickListener {
            launchFreeCatPicture()
        }
    }

    fun launchFreeCatPicture() {
        val intent = Intent(this, FreePictureActivity::class.java)
        startActivity(intent)
    }

}
```

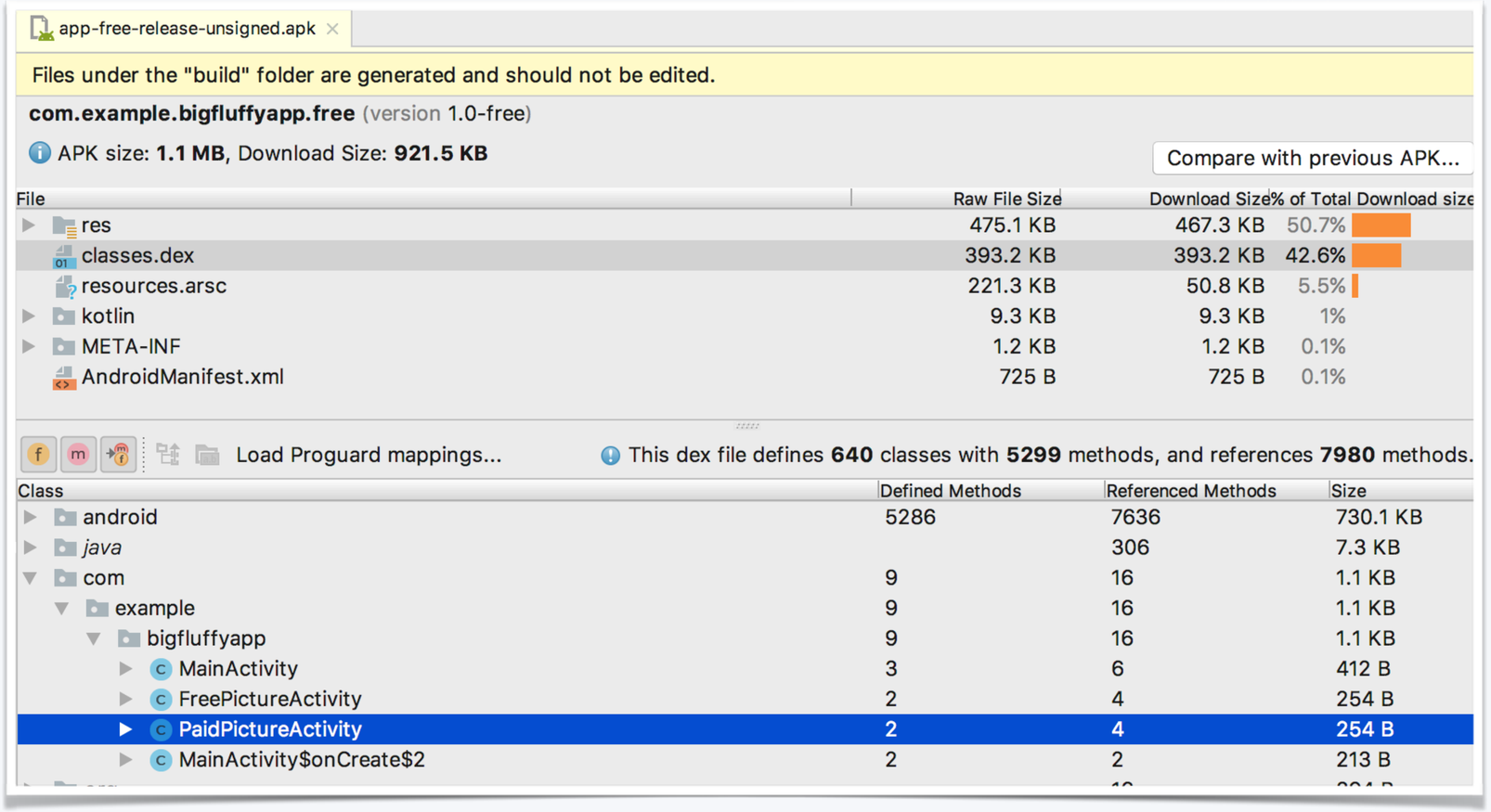
Back to the Analyzer



Back to the Analyzer

```
const v0, 0x7f070026

.line 19
invoke-virtual {p0, v0}, Lcom/example/bigfluffyapp/MainActivity;->findViewById(I)Landroid/view/View;

move-result-object v0

const-string v1, "findViewById(R.id.click_for_another_cat_picture)"

invoke-static {v0, v1}, Lkotlin/jvm/internal/Intrinsics;->checkExpressionValueIsNotNull(Ljava/lang/Object;Ljava/lang/String;)V

check-cast v0, Landroid/widget/Button;

const/16 v1, 0x8

.line 29
invoke-virtual {v0, v1}, Landroid/widget/Button;->setVisibility(I)V
```

Back to the Analyzer

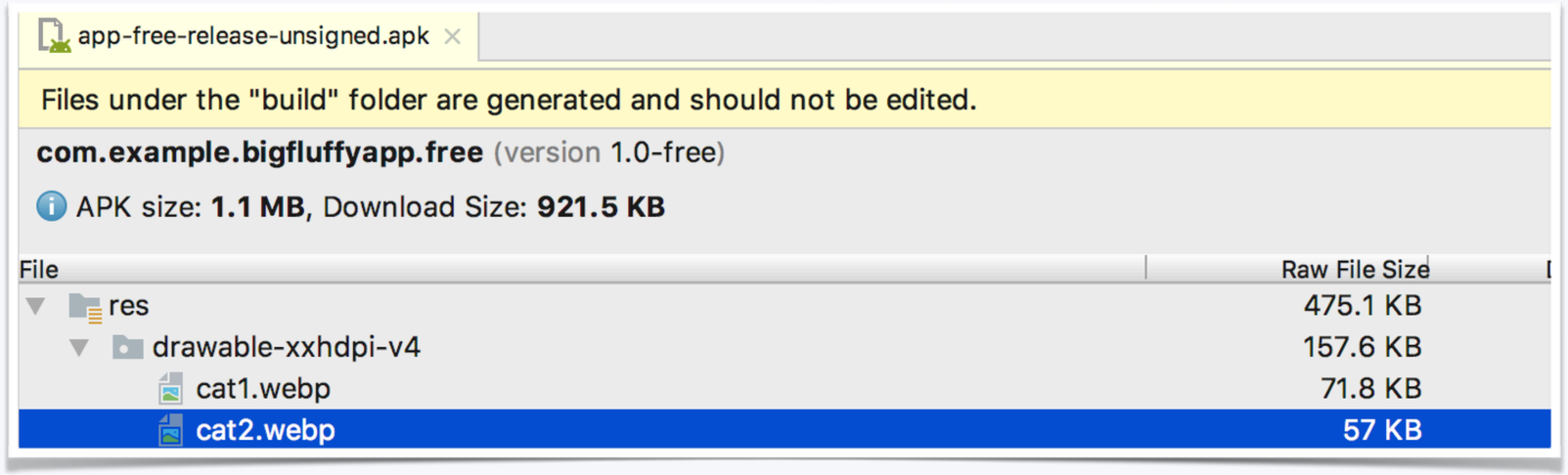


The unused, unlaunched activity **did not get stripped by ProGuard** 🤔

The “Paid” Activity

```
class PaidPictureActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_paid)

        val imageView = findViewById<ImageView>(R.id.imageView)
        imageView.setBackgroundResource(R.drawable.cat2)

    }
}
```

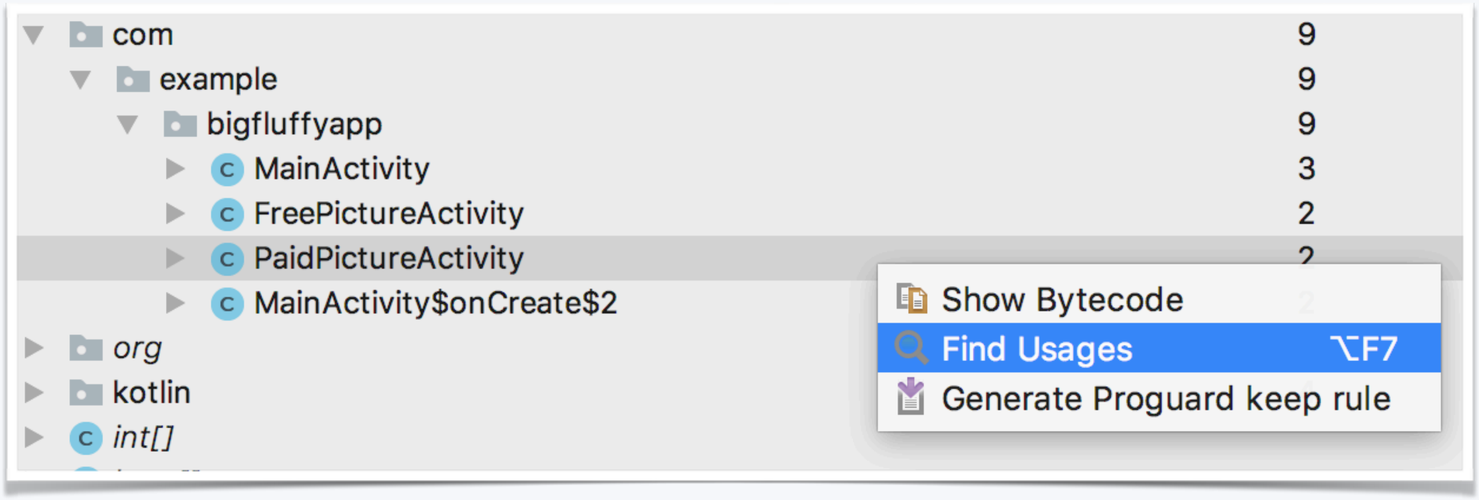
Back to the Analyzer



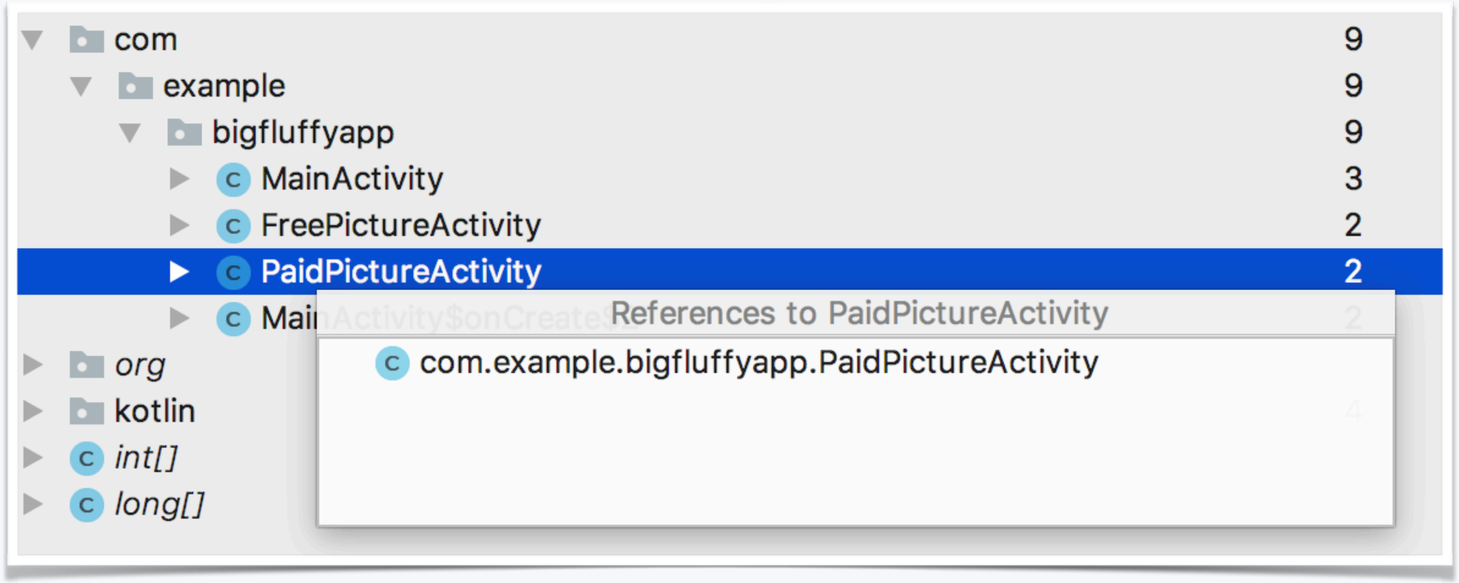
But the activity is still there

and so is a resource it refers to

Back to the Analyzer



Back to the Analyzer



AAPT's ProGuard Tricks

AAPT, the *Android Asset Packaging Tool*, **auto-generates ProGuard rules for classes referenced in your XML**

Check out **`aapt_rules.txt`** in your build's intermediates path

```
# Referenced at /Users/ben/git/BigFluffyApp/app/build/intermediates/manifests/full/free/release/AndroidManifest.xml:26
-keep class com.example.bigfluffyapp.PaidPictureActivity { <init>(...); }
```

It will **tell you why it put the ProGuard rule there**

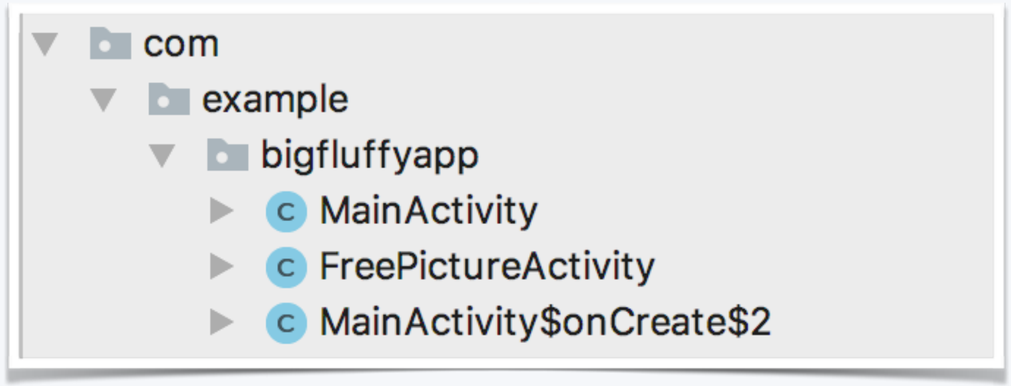
Giving AAPT a nudge

Let’s create a “free” flavor manifest that **removes the PaidPictureActivity**

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <application>
        <activity android:name="com.example.bigfluffyapp.PaidPictureActivity"
            tools:node="remove"/>
    </application>

</manifest>
```

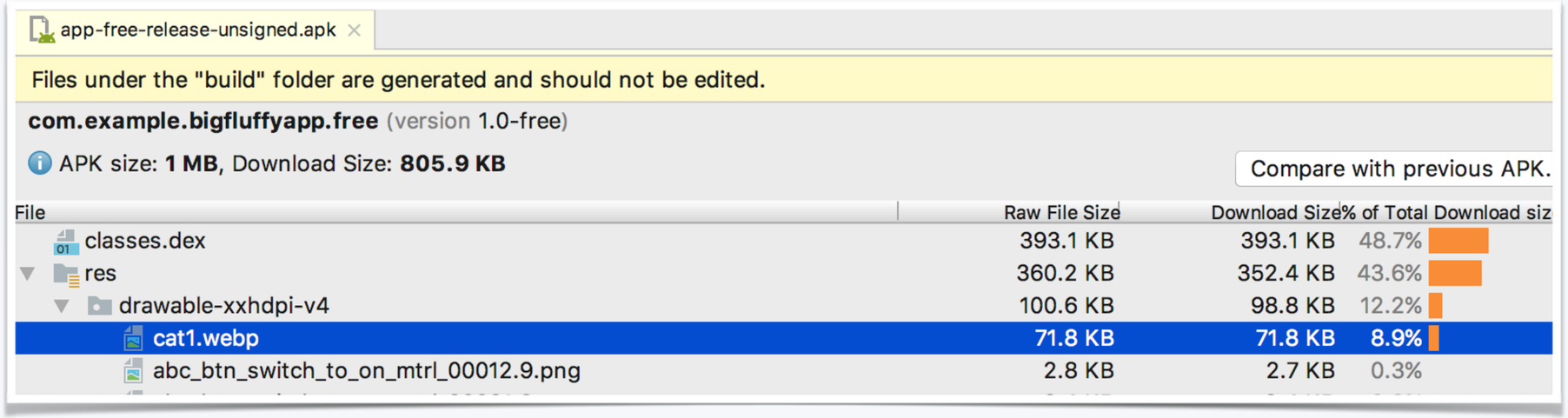
The remove directive **hints the manifest merger to remove this activity**

Now How'd We Do?



Success! The unused activity is gone

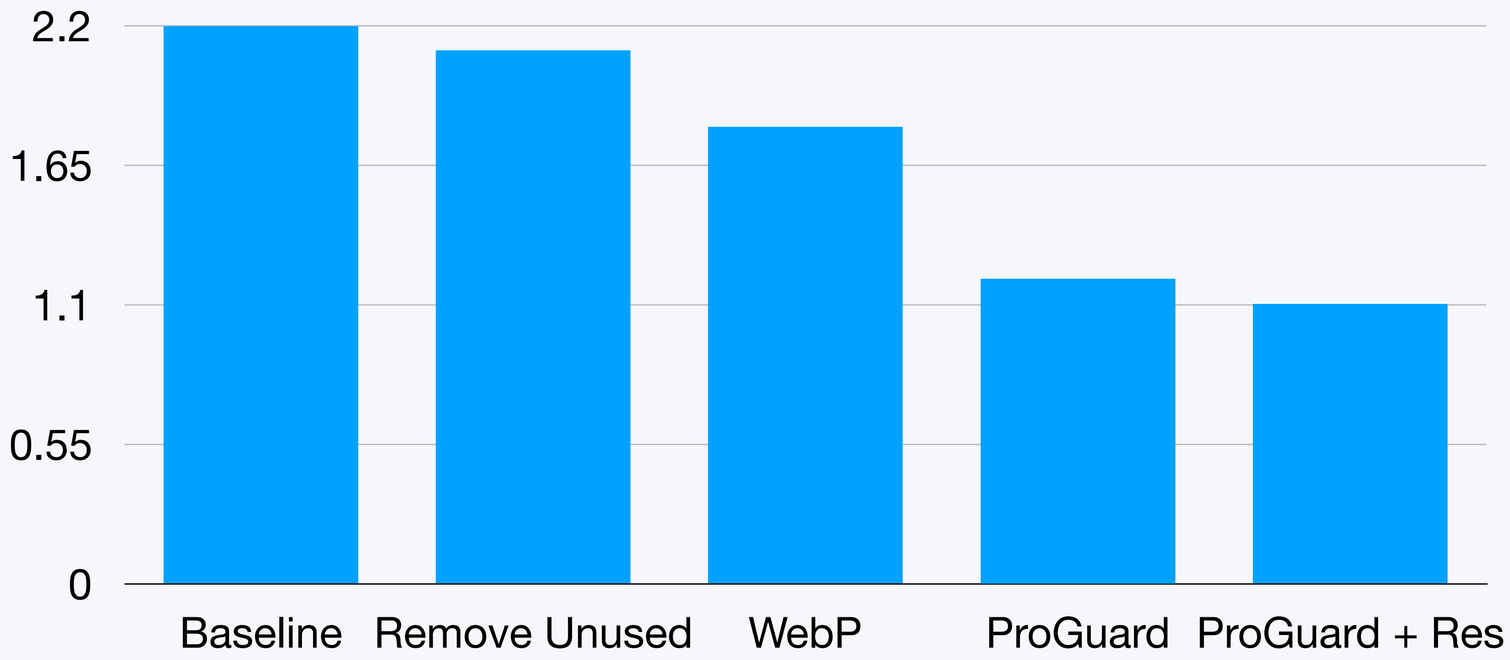
Now How'd We Do?



And so is the resource it would've referred to.

Now How'd We Do?

With unused code and resources stripped from the free variant, cumulatively we've saved 1.1MB relative to baseline.



APK Splits

What If...

We could make **separate APK files for given screen densities and binary architectures?**

We'd be able to shrink each APK **by delivering only necessary resources**

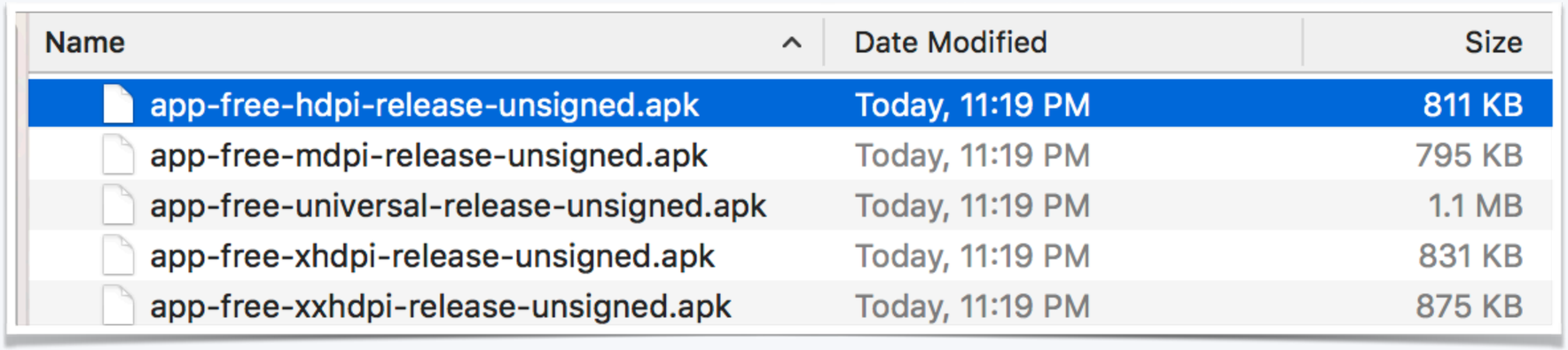
APK Splits

```
android {

    // ...

    splits {

        // Configures multiple APKs based on screen density.
        density {

            // Configures multiple APKs based on screen density.
            enable true

            // Specifies a list of compatible screen size settings for the manifest.
            compatibleScreens 'small', 'normal', 'large', 'xlarge'
        }

    }
}
```

APK Splits

```
android {

    // ...

    abi {

        // Enables building multiple APKs per ABI.
        enable true

        // By default all ABIs are included, so use reset() and include to specify that we only
        // want APKs for x86, armeabi-v7a, and mips.

        // Resets the list of ABIs that Gradle should create APKs for to none.
        reset()

        // Specifies a list of ABIs that Gradle should create APKs for.
        include "x86", "armeabi-v7a", "mips"

        // Specifies that we do not want to also generate a universal APK that includes all ABIs.
        universalApk false
    }
}
```

APK Splits



APKs for each **display density!**

Caveat Emptor

This generates **multiple APK files with the same version code**

The Play Store doesn't like that so we have to be creative with the version code

Devices will get the APK with **the highest version code that matches their device's architecture/display density**

Version Codes

We'll steal the 10000's place in the version code **to denote the density**

For a build with version code 100:

Density	Version Code
mdpi	10100
hdpi	20100
xhdpi	30100
xxhdpi	30100
xxxhdpi	40100

Version Code

```
ext.densityCodes = ['mdpi':1, 'hdpi':2, 'xhdpi':3, 'xxhdpi':4, 'xxxhdpi': 5]

import com.android.build.OutputFile

android.applicationVariants.all { variant ->

    // Assigns a different version code for each output APK
    // other than the universal APK.
    variant.outputs.each { output ->

        def baseDensityVersionCode =
                project.ext.densityCodes.get(output.getFilter(OutputFile.DENSITY))

        // for a universal APK this value will end up null, which is good, because
        // we want the universal to be the lowest version code -- this way the
        // play store treats it as a fallback
        if (baseDensityVersionCode != null) {

            //Assign the new versionCode
            output.versionCodeOverride = (baseDensityVersionCode * 10000) + variant.versionCode
        }
    }
}
```

Now How'd We Do?

For the xxhdpi version of the free APK (Google Pixel) we've saved 1.4MB relative to baseline.