Androguard Documentation

Release 3.4.0

Anthony Desnos

Oct 18, 2019

CONTENTS

Androguard is a full python tool to play with Android files. It is designed to work with Python 3 only.

- DEX, ODEX
- APK
- Android’s binary xml
- Android resources
- Disassemble DEX/ODEX bytecodes
- Decompiler for DEX/ODEX files

You can either use the cli or graphical frontend for androguard, or use androguard purely as a library for your own tools and scripts.

CHAPTER

ONE

DOCUMENTATION

1.1 Introduction

1.1.1 Installation

There are several ways how to install androguard.

Before you start, make sure you are using a supported python version! For Windows, we recommend using the Anaconda python 3.6.x package.

Warning: The magic library might not work out of the box. If your magic library does not work, please refer to the installation instructions of python-magic.

PIP

The usual way to install a python packages is by using pypi.python.org and it's package installer *pip*. Just use

```
$ pip install -U androguard[magic,GUI]
```

to install androguard including the GUI and magic file type detection. In order to use features which use `dot`, you need Graphviz installed. This is not a python dependency but a binary package! Please follow the installation instructions for GraphvizInstall.

You can also make use of an *virtualenv*, to separate the installation from your system wide packages:

```
$ virtualenv venv-androguard
$ source venv-androguard/bin/activate
$ pip install -U androguard[magic,GUI]
```

pip should install all required packages too.

Debian / Ubuntu

Debian has androguard in its repository. You can just install it using `apt install androguard`. All required dependencies are automatically installed.

Install from Source

Use git to fetch the sources, then install it. Please install git and python on your own. Androguard requires Python at least 3.4 to work. Pypy >= 5.9.0 should work as well but is not tested.

```
$ git clone --recursive https://github.com/androguard/androguard.git
$ cd androguard
$ virtualenv -p python3 venv-androguard
$ source venv-androguard/bin/activate
$ pip install .[magic,GUI]
```

The dependencies, defined in `setup.py` will be automatically installed.

For development purposes, you might want to install the extra dependecies for *docs* and *tests* as well:

```
$ git clone --recursive https://github.com/androguard/androguard.git
$ cd androguard
$ virtualenv -p python3 venv-androguard
$ source venv-androguard/bin/activate
$ pip install -e .[magic,GUI,tests,docs]
```

You can then create a local copy of the documentation:

```
$ python3 setup.py build_sphinx
```

Which is generated in `build/sphinx/html`.

1.1.2 Getting Started

Using Androguard tools

There are already some tools for specific purposes.

To just decode the AndroidManifest.xml or resources.arsc, there are *androguard axml* and *androguard arsc*. To get information about the certificates use *androguard sign*.

If you want to create call graphs, use *androguard cg*, or if you want control flow graphs, you can use *androguard decompile*.

Using Androlyze and the python API

The easiest way to analyze APK files, is by using `androguard analyze`. It will start a iPython shell and has all modules loaded to get into action.

For analyzing and loading APK or DEX files, some wrapper functions exists. Use `AnalyzeAPK(filename)` or `AnalyzeDEX(filename)` to load a file and start analyzing. There are already plenty of APKs in the androguard repo, you can either use one of those, or start your own analysis.

```
$ androguard analyze
Androguard version 3.1.1 started
In [1]: a, d, dx = AnalyzeAPK("examples/android/abcore/app-prod-debug.apk")
# Depending on the size of the APK, this might take a while...

In [2]:
```

The three objects you get are a an *APK* object, d an array of *DalvikVMFormat* object and dx an *Analysis* object.

Inside the APK object, you can find all information about the APK, like package name, permissions, the AndroidManifest.xml or its resources.

The *DalvikVMFormat* corresponds to the DEX file found inside the APK file. You can get classes, methods or strings from the DEX file. But when using multi-DEX APK's it might be a better idea to get those from another place. The *Analysis* object should be used instead, as it contains special classes, which link information about the classes.dex and can even handle many DEX files at once.

Getting Information about an APK

If you have sucessfully loaded your APK using AnalyzeAPK, you can now start getting information about the APK.

For example, getting the permissions of the APK:

```
In [2]: a.get_permissions()
Out[2]:
['android.permission.INTERNET',
 'android.permission.WRITE_EXTERNAL_STORAGE',
 'android.permission.ACCESS_WIFI_STATE',
 'android.permission.ACCESS_NETWORK_STATE']
```

or getting a list of all activites, which are defined in the AndroidManifest.xml:

```
In [3]: a.get_activities()
Out[3]:
['com.greenaddress.abcore.MainActivity',
 'com.greenaddress.abcore.BitcoinConfEditActivity',
 'com.greenaddress.abcore.AboutActivity',
 'com.greenaddress.abcore.SettingsActivity',
 'com.greenaddress.abcore.DownloadSettingsActivity',
 'com.greenaddress.abcore.PeerActivity',
 'com.greenaddress.abcore.ProgressActivity',
 'com.greenaddress.abcore.LogActivity',
 'com.greenaddress.abcore.ConsoleActivity',
 'com.greenaddress.abcore.DownloadActivity']
```

Get the package name, app name and path of the icon:

```
In [4]: a.get_package()
Out[4]: 'com.greenaddress.abcore'

In [5]: a.get_app_name()
Out[5]: u'ABCore'

In [6]: a.get_app_icon()
Out[6]: u'res/mipmap-xxxhdpi-v4/ic_launcher.png'
```

Get the numeric version and the version string, and the minimal, maximal, target and effective SDK version:

```
In [7]: a.get_androidversion_code()
Out[7]: '2162'

In [8]: a.get_androidversion_name()
Out[8]: '0.62'

```

(continues on next page)

(continued from previous page)

```
In [9]: a.get_min_sdk_version()
Out[9]: '21'

In [10]: a.get_max_sdk_version()

In [11]: a.get_target_sdk_version()
Out[11]: '27'

In [12]: a.get_effective_target_sdk_version()
Out[12]: 27
```

You can even get the decoded XML for the AndroidManifest.xml:

```
In [15]: a.get_android_manifest_axml().get_xml()
Out[15]: '<manifest xmlns:android="http://schemas.android.com/apk/res/android"␣
↪android:versionCode="2162" android:versionName="0.62" package="com.greenaddress.
↪abcore">\n<uses-sdk android:minSdkVersion="21" android:targetSdkVersion="27">\n</
↪uses-sdk>\n<uses-permission android:name="android.permission.INTERNET">\n</uses-
↪permission>\n<uses-permission android:name="android.permission.WRITE_EXTERNAL_
↪STORAGE">\n</uses-permission>\n<uses-permission android:name="android.permission.
↪ACCESS_WIFI_STATE">\n</uses-permission>\n<uses-permission android:name="android.
↪permission.ACCESS_NETWORK_STATE">\n</uses-permission>\n<application android:theme=
↪"@7F0F0006" android:label="@7F0E001D" android:icon="@7F0D0000" android:debuggable=
↪"true" android:allowBackup="false" android:supportsRtl="true">\n<activity␣
↪android:name="com.greenaddress.abcore.MainActivity">\n<intent-filter>\n<action␣
↪android:name="android.intent.action.MAIN">\n</action>\n<category android:name=
↪"android.intent.category.LAUNCHER">\n</category>\n</intent-filter>\n</activity>\n
↪<service android:name="com.greenaddress.abcore.DownloadInstallCoreIntentService"␣
↪android:exported="false">\n</service>\n<service android:name="com.greenaddress.
↪abcore.RPCIntentService" android:exported="false">\n</service>\n<service␣
↪android:name="com.greenaddress.abcore.ABCoreService" android:exported="false">\n</
↪service>\n<activity android:name="com.greenaddress.abcore.BitcoinConfEditActivity">
↪\n<intent-filter>\n<category android:name="android.intent.category.DEFAULT">\n</
↪category>\n<action android:name="com.greenaddress.abcore.BitcoinConfEditActivity">\n
↪</action>\n</intent-filter>\n</activity>\n<activity android:name="com.greenaddress.
↪abcore.AboutActivity">\n</activity>\n<activity android:label="@7F0E0038"␣
↪android:name="com.greenaddress.abcore.SettingsActivity" android:noHistory="true">\n
↪</activity>\n<activity android:label="@7F0E0035" android:name="com.greenaddress.
↪abcore.DownloadSettingsActivity" android:noHistory="true">\n</activity>\n<activity␣
↪android:theme="@7F0F0006" android:label="@7F0E0036" android:name="com.greenaddress.
↪abcore.PeerActivity">\n</activity>\n<activity android:theme="@7F0F0006"␣
↪android:label="@7F0E0037" android:name="com.greenaddress.abcore.ProgressActivity">\n
↪</activity>\n<activity android:name="com.greenaddress.abcore.LogActivity">\n</
↪activity>\n<activity android:name="com.greenaddress.abcore.ConsoleActivity">\n</
↪activity>\n<activity android:name="com.greenaddress.abcore.DownloadActivity">\n</
↪activity>\n<receiver android:name="com.greenaddress.abcore.PowerBroadcastReceiver">
↪\n<intent-filter>\n<action android:name="android.intent.action.ACTION_POWER_
↪CONNECTED">\n</action>\n<action android:name="android.intent.action.ACTION_POWER_
↪DISCONNECTED">\n</action>\n<action android:name="android.intent.action.ACTION_
↪SHUTDOWN">\n</action>\n<action android:name="android.intent.action.ACTION_BATTERY_
↪LOW">\n</action>\n<action android:name="android.net.wifi.STATE_CHANGE">\n</action>\n
↪</intent-filter>\n</receiver>\n</application>\n</manifest>\n'
```

Or if you like to use the AndroidManifest.xml as an ElementTree object, use the following method:

```
In [13]: a.get_android_manifest_xml()
Out[13]: <Element manifest at 0x7f9d01587b00>
```

There are many more methods to explore, just take a look at the API for `APK`.

Using the Analysis object

The `~androguard.core.analysis.analysis.Analysis` object has all information about the classes, methods, fields and strings inside one or multiple DEX files.

Additionally it enables you to get call graphs and crossreferences (XREFs) for each method, class, field and string.

This means you can investigate the application for certain API calls or create graphs to see the dependencies of different classes.

As a first example, we will get all classes from the Analysis:

```
In [2]: dx.get_classes()
Out[2]:
[<analysis.ClassAnalysis Ljava/io/FileNotFoundException; EXTERNAL>,
 <analysis.ClassAnalysis Landroid/content/SharedPreferences; EXTERNAL>,
 <analysis.ClassAnalysis Landroid/support/v4/widget/FocusStrategy$BoundsAdapter;>,
 <analysis.ClassAnalysis Landroid/support/v4/media/MediaBrowserCompat
↪$MediaBrowserServiceCallbackImpl;>,
 <analysis.ClassAnalysis Landroid/support/transition/WindowIdImpl;>,
 <analysis.ClassAnalysis Landroid/media/MediaMetadataEditor; EXTERNAL>,
 <analysis.ClassAnalysis Landroid/support/v4/app/BundleCompat$BundleCompatBaseImpl;>,
 <analysis.ClassAnalysis Landroid/support/transition/MatrixUtils$1;>,
 <analysis.ClassAnalysis Landroid/support/v7/widget/ShareActionProvider;>,
 ...
```

As you can see, `get_classes()` returns a list of `ClassAnalysis` objects. Some of them are marked as *EXTERNAL*, which means that the source code of this class is not defined within the DEX files that are loaded inside the Analysis. For example the first class `java.io.FileNotFoundException` is an API class.

A `ClassAnalysis` does not contain the actual code but the `ClassDefItem` can be loaded using the `get_vm_class()`:

```
In [5]: dx.get_classes()[2].get_vm_class()
Out[5]: <dvm.ClassDefItem Ljava/lang/Object;->Landroid/support/v4/widget/FocusStrategy
↪$BoundsAdapter;>
```

If the class is *EXTERNAL*, a `ExternalClass` is returned instead.

The `ClassAnalysis` also contains all the information about XREFs, which are explained in more detail in the next section.

XREFs

Consider the following Java source code:

```
class Foobar {
    public int afield = 23;

    public void somemethod() {
        String astring = "hello world";
    }
}

class Barfoo {
```

(continues on next page)

(continued from previous page)

```
    public void othermethod() {
        Foobar x = new Foobar();

        x.somemethod();

        System.out.println(x.afield);
    }
}
```

There are two classes and the class `Barfoo` instanciates the other class `Foobar` as well as calling methods and reading fields.

XREFs are generated for four things:

- Classes
- Methods
- Fields
- Strings

XREFs work in two directions: *xref_from* and *xref_to*. *To* means, that the current object is calling another object. *From* means, that the current object is called by another object.

All XREFs can be visualized as an directed graph and if some object `A` is contained in the `xref_to`, the called object will contain `A` in their `xref_from`.

In the case of our Java example, the string `astring` is called in `Foobar.somethod`, therefore it will be contained in the `xref_to` of `Foobar.somethod`.

The Field `afield` will be contained in the `xref_to` of `Barfoo.othermethod` as well as the call to `Foobar.somethod`.

More on XREFs can be found in *xrefs*.

1.1.3 Crossreferences (XREFs)

Crossreferences or simply XREFs are the main thing which *`Analysis`* provides. XREFs are generated for Classes, Methods, Fields and Strings.

Next, we want to show a few usecases for XREFs and how they can be obtained.

Start up a ipython shell using `androguard analyze` in order to play through the example. We use an example from the androguard repo here:

```
$ androguard analyze examples/android/TestsAndroguard/bin/TestActivity.apk
Please be patient, this might take a while.
Found the provided file is of type 'APK'
[INFO    ] androguard.analysis: End of creating cross references (XREF)
[INFO    ] androguard.analysis: run time: 0min 00s
Added file to session:␣
↪SHA256::3bb32dd50129690bce850124ea120aa334e708eaa7987cf2329fd1ea0467a0eb
Loaded APK file...
>>> a
<androguard.core.bytecodes.apk.APK object at 0x000000000581D710>
>>> d
[<androguard.core.bytecodes.dvm.DalvikVMFormat object at 0x000000000D847400>]
>>> dx
```

(continues on next page)

(continued from previous page)

```
<analysis.Analysis VMs: 1, Classes: 495, Strings: 496>

Androguard version 3.3.5 started
In [1]:
```

Get XREFs for method calls

The first example would be to query all called classes from the class `tests.androguard.TestActivity`. Remember, that you need to provide the class name as a type format with forward slashes instead of dots. In order to get the class, you can simply use `classes` or *`find_classes()`*:

```
In [4]: dx.classes['Ltests/androguard/TestActivity;']
Out[4]: <analysis.ClassAnalysis Ltests/androguard/TestActivity;>
```

This will return a *`ClassAnalysis`* object. Now you can iterate over all methods inside the class and query for the xrefs (the output is abbreviated):

```
In [10]: for meth in dx.classes['Ltests/androguard/TestActivity;'].get_methods():
    ...:     print("inside method {}".format(meth.name))
    ...:     for _, call, _ in meth.get_xref_to():
    ...:         print("  calling -> {} -- {}".format(call.class_name, call.name))
    ...:
inside method testCall1
  calling -> Ljava/lang/StringBuilder; -- toString
  calling -> Ljava/lang/StringBuilder; -- append
  calling -> Ljava/lang/StringBuilder; -- <init>
  calling -> Ljava/io/PrintStream; -- println
inside method testCalls
  calling -> Ljava/lang/Object; -- getClass
  calling -> Ljava/io/PrintStream; -- println
  calling -> Ltests/androguard/TestIfs; -- testIF
  calling -> Ltests/androguard/TestActivity; -- testCall2
[...]
```

Here you can see, that `tests.androguard.TestActivity.testCall1` uses a `StringBuilder` as well as `println`. The method `testCalls` is calling other functions from the same package.

The other way around is also possible. Especially for Android API's, this is very interesting!

Note: External method, like the API calls, will not give any XREFs for `xref_to()`.

Lets say, you want all calls to the API class `java.io.file`:

```
In [3]: dx.classes['Ljava/io/File;']
Out[3]: <analysis.ClassAnalysis Ljava/io/File; EXTERNAL>

In [4]: for meth in dx.classes['Ljava/io/File;'].get_methods():
   ...:     print("usage of method {}".format(meth.name))
   ...:     for _, call, _ in meth.get_xref_from():
   ...:         print("  called by -> {} -- {}".format(call.class_name, call.name))
   ...:
usage of method getPath
  called by -> Landroid/support/v4/util/AtomicFile; -- <init>
usage of method <init>
```

(continues on next page)

(continued from previous page)

```
  called by -> Landroid/support/v4/util/AtomicFile; -- <init>
usage of method delete
  called by -> Landroid/support/v4/util/AtomicFile; -- failWrite
  called by -> Landroid/support/v4/util/AtomicFile; -- delete
  called by -> Landroid/support/v4/util/AtomicFile; -- delete
  called by -> Landroid/support/v4/util/AtomicFile; -- startWrite
  called by -> Landroid/support/v4/util/AtomicFile; -- openRead
  called by -> Landroid/support/v4/util/AtomicFile; -- finishWrite
usage of method renameTo
  called by -> Landroid/support/v4/util/AtomicFile; -- openRead
  called by -> Landroid/support/v4/util/AtomicFile; -- failWrite
  called by -> Landroid/support/v4/util/AtomicFile; -- startWrite
usage of method exists
  called by -> Landroid/support/v4/util/AtomicFile; -- startWrite
  called by -> Landroid/support/v4/util/AtomicFile; -- openRead
  called by -> Landroid/support/v4/util/AtomicFile; -- startWrite
usage of method getParentFile
  called by -> Landroid/support/v4/util/AtomicFile; -- startWrite
usage of method mkdir
  called by -> Landroid/support/v4/util/AtomicFile; -- startWrite
```

Note: An external class or method is simply a class or method which could not be found inside the loaded DEX files at the time the XREFs were created! Thus, it is important to always load all DEX files of a multidex file. On the other hand, beware that classes might not be defined as they could be loaded dynamically later. External does not automatically mean that this class/method is an Android or Java API!

Get XREFs for Strings

Next, we want to see where certain strings are used. For example, you found the interesting String `'boom'` and would like to know where it is used. You can use either `strings` or *`find_strings()`* to get the proper object for the XREFs:

```
In [12]: dx.strings['boom']
Out[12]: <analysis.StringAnalysis 'boom'>
```

The resulting object is of type *`StringAnalysis`*.

Note: *`StringAnalysis`* does not have a `xref_to` method, which is obvious, as a String does nothing but is always used.

Now we can call `xref_from()` to get the usage of the String:

```
In [14]: for _, meth in dx.strings['boom'].get_xref_from():
    ...:     print("Used in: {} -- {}".format(meth.class_name, meth.name))
    ...:
Used in: Ltests/androguard/TestActivity; -- test_base
```

So, we know that this specific String is used once in the `test_base` method.

Get XREFs for Fields

The last XREF we can use are fields. Fields are a little bit different and do not use `xref_from` and `xref_to` but `xref_read()` and `xref_write()`. You can use the method *`find_methods()`* in order to find fields.

Note: Calls to static fields are usually not tracked, as they are optimized by the compiler to const calls!

For example, you want to get the read's and write's to the field `value` inside `tests.androguard.TestActivity`:

```
In [25]: for field in dx.find_fields(classname='Ltests/androguard/TestActivity;',␣
↪fieldname='^value$'):
    ...:     print("Field: {}".format(field.name))
    ...:     for _, meth in field.get_xref_read():
    ...:         print("  read in {} -- {}".format(meth.class_name, meth.name))
    ...:     for _, meth in field.get_xref_write():
    ...:         print("  write in {} -- {}".format(meth.class_name, meth.name))
    ...:
Field: value
  read in Ltests/androguard/TestActivity; -- pouet
  read in Ltests/androguard/TestActivity; -- test1
  read in Ltests/androguard/TestActivity; -- test_base
  read in Ltests/androguard/TestActivity; -- testVars
  write in Ltests/androguard/TestActivity; -- <init>
  write in Ltests/androguard/TestActivity; -- pouet2
  write in Ltests/androguard/TestActivity; -- <init>
  write in Ltests/androguard/TestActivity; -- <init>
```

1.1.4 Basic Blocks

We already saw the concept of **xrefs_**, which can be used to get references in the assembly. The next step is to look at the Control Flow Graph (CFG) of a method.

Such a CFG can be generated using the *decompile* command of the **androguard_** tool. Let's take the androguard example file and decompile it:

```
$ androguard decompile -d output_folder -f jpg --limit "LTestDefaultPackage.*"␣
↪examples/android/TestsAndroguard/bin/TestActivity.apk
[INFO    ] androguard.analysis: End of creating cross references (XREF)
[INFO    ] androguard.analysis: run time: 0min 00s
Dump information examples/android/TestsAndroguard/bin/TestActivity.apk in output_
↪folder
Create directory output_folder
Decompilation ... End
Dump LTestDefaultPackage$TestInnerClass$TestInnerInnerClass; <init>␣
↪(LTestDefaultPackage$TestInnerClass; I I)V ... jpg ... source codes ... bytecodes ..
↪.
Dump LTestDefaultPackage$TestInnerClass$TestInnerInnerClass; <init>␣
↪(LTestDefaultPackage$TestInnerClass; I I LTestDefaultPackage$TestInnerClass
↪$TestInnerInnerClass;)V ... jpg ... bytecodes ...
Dump LTestDefaultPackage$TestInnerClass$TestInnerInnerClass; Test (I)V ... jpg ...␣
↪bytecodes ...
Dump LTestDefaultPackage$TestInnerClass; <init> (LTestDefaultPackage; I I)V ... jpg ..
↪. source codes ... bytecodes ...
Dump LTestDefaultPackage$TestInnerClass; <init> (LTestDefaultPackage; I I␣
↪LTestDefaultPackage$TestInnerClass;)V ... jpg ... bytecodes ...
```

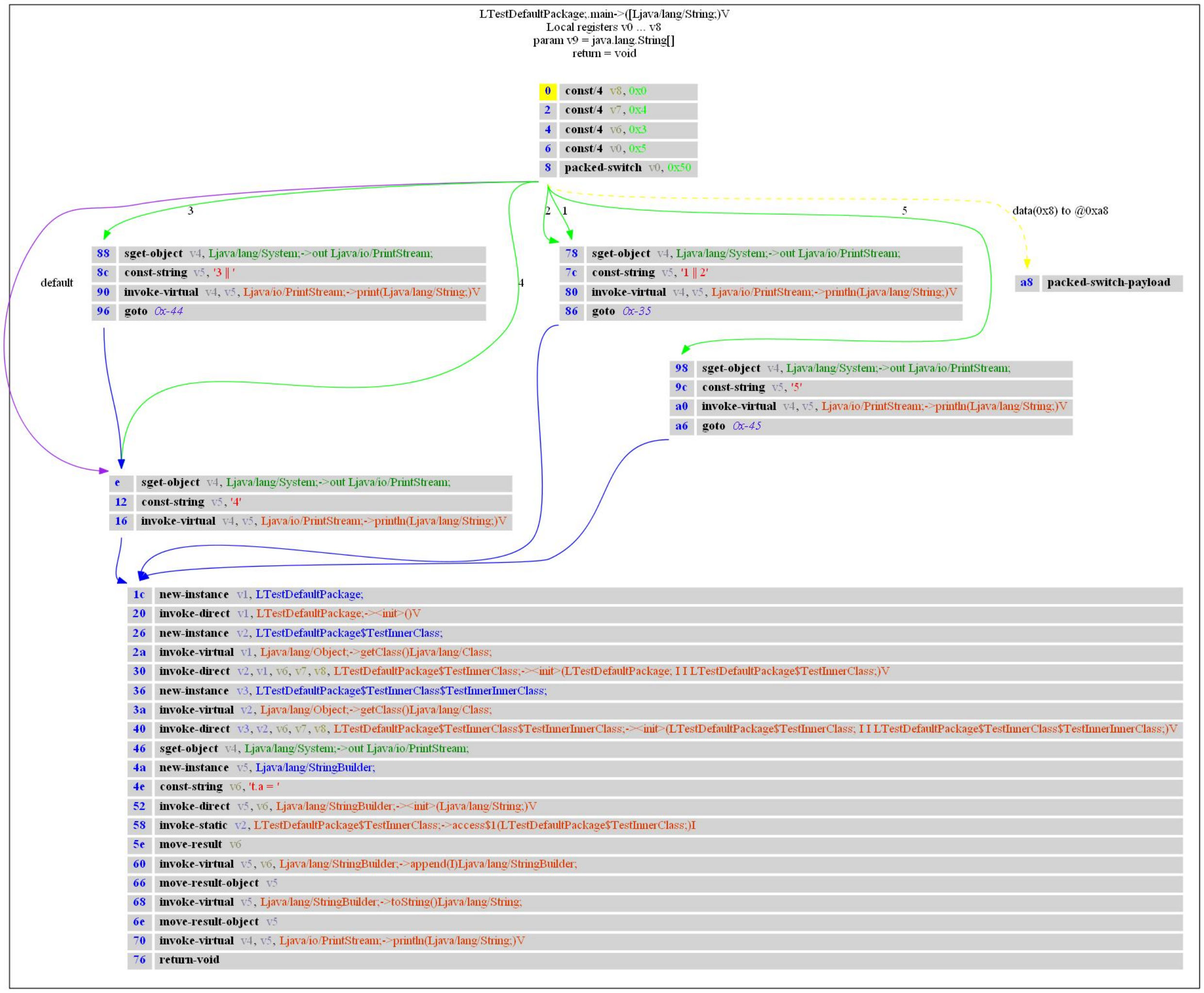
(continues on next page)

(continued from previous page)

```
Dump LTestDefaultPackage$TestInnerClass; access$1 (LTestDefaultPackage$TestInnerClass;
↪)I ... jpg ... bytecodes ...
Dump LTestDefaultPackage$TestInnerClass; Test (I)V ... jpg ... bytecodes ...
Dump LTestDefaultPackage; <init> ()V ... jpg ... source codes ... bytecodes ...
Dump LTestDefaultPackage; main ([Ljava/lang/String;)V ... jpg ... bytecodes ...
Dump LTestDefaultPackage; const4 ()V ... jpg ... bytecodes ...
```

Note, that we only decompiled a certain subset of the file, as we are not interested in the other classes right now.

Inside the output folder, we have now several files, among them some JPG files which show the CFG, like this one:



```
LTestDefaultPackage;.main->([Ljava/lang/String;)V
Local registers v0 ... v8
param v9 = java.lang.String[]
return = void

0   const/4  v8, 0x0
2   const/4  v7, 0x4
4   const/4  v6, 0x3
6   const/4  v0, 0x5
8   packed-switch  v0, 0x50

88  sget-object  v4, Ljava/lang/System;->out Ljava/io/PrintStream;
8c  const-string  v5, '3 || '
90  invoke-virtual  v4, v5, Ljava/io/PrintStream;->print(Ljava/lang/String;)V
96  goto  0x-44

78  sget-object  v4, Ljava/lang/System;->out Ljava/io/PrintStream;
7c  const-string  v5, '1 || 2'
80  invoke-virtual  v4, v5, Ljava/io/PrintStream;->println(Ljava/lang/String;)V
86  goto  0x-35

a8  packed-switch-payload

98  sget-object  v4, Ljava/lang/System;->out Ljava/io/PrintStream;
9c  const-string  v5, '5'
a0  invoke-virtual  v4, v5, Ljava/io/PrintStream;->println(Ljava/lang/String;)V
a6  goto  0x-45

e   sget-object  v4, Ljava/lang/System;->out Ljava/io/PrintStream;
12  const-string  v5, '4'
16  invoke-virtual  v4, v5, Ljava/io/PrintStream;->println(Ljava/lang/String;)V

1c  new-instance  v1, LTestDefaultPackage;
20  invoke-direct  v1, LTestDefaultPackage;-><init>()V
26  new-instance  v2, LTestDefaultPackage$TestInnerClass;
2a  invoke-virtual  v1, Ljava/lang/Object;->getClass()Ljava/lang/Class;
30  invoke-direct  v2, v1, v6, v7, v8, LTestDefaultPackage$TestInnerClass;-><init>(LTestDefaultPackage; I I LTestDefaultPackage$TestInnerClass;)V
36  new-instance  v3, LTestDefaultPackage$TestInnerClass$TestInnerInnerClass;
3a  invoke-virtual  v2, Ljava/lang/Object;->getClass()Ljava/lang/Class;
40  invoke-direct  v3, v2, v6, v7, v8, LTestDefaultPackage$TestInnerClass$TestInnerInnerClass;-><init>(LTestDefaultPackage$TestInnerClass; I I LTestDefaultPackage$TestInnerClass$TestInnerInnerClass;)V
46  sget-object  v4, Ljava/lang/System;->out Ljava/io/PrintStream;
4a  new-instance  v5, Ljava/lang/StringBuilder;
4e  const-string  v6, 't.a = '
52  invoke-direct  v5, v6, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V
58  invoke-static  v2, LTestDefaultPackage$TestInnerClass;->access$1(LTestDefaultPackage$TestInnerClass;)I
5e  move-result  v6
60  invoke-virtual  v5, v6, Ljava/lang/StringBuilder;->append(I)Ljava/lang/StringBuilder;
66  move-result-object  v5
68  invoke-virtual  v5, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;
6e  move-result-object  v5
70  invoke-virtual  v4, v5, Ljava/io/PrintStream;->println(Ljava/lang/String;)V
76  return-void
```

Each of the rectangles is a `DVMBasicBlock`. Each block is connected via an arrow, indicating the flow direction.

In this example, we can see that the *switch* instruction has six different ways to go, indicated by the green and purple arrows. Each green arrow is a specific check inside the *switch* instruction, i.e. what value results in which code block. The purple arrow is the default case. We can see that the *switch* only results in four different code blocks. There is a special block, with the yellow arrow, which is the pseudo instruction holding the switch payload.

Each of the switch blocks is followed by another, large basic block. If you look carefully, you can see that three of the blocks have *goto* commands at the end but the fourth block does not have one. First, take a look at the overall dissassembly of the method:

```
METHOD LTestDefaultPackage; public static main ([Ljava/lang/String; v9)V
main-BB@0x00000000 :
        0   (00000000) const/4             v8, 0
        1   (00000002) const/4             v7, 4
        2   (00000004) const/4             v6, 3
        3   (00000006) const/4             v0, 5
        4   (00000008) packed-switch       v0, 80 [ D:main-BB@0x0000000e 1:main-
↪BB@0x00000078 2:main-BB@0x00000078 3:main-BB@0x00000088 4:main-BB@0x0000000e 5:main-
↪BB@0x00000098 ]
        5   (0000000e) sget-object         v4, Ljava/lang/System;->out Ljava/io/
↪PrintStream;
        6   (00000012) const-string        v5, '4'
        7   (00000016) invoke-virtual      v4, v5, Ljava/io/PrintStream;->
↪println(Ljava/lang/String;)V [ main-BB@0x0000001c ]
        8   (0000001c) new-instance        v1, LTestDefaultPackage;
        9   (00000020) invoke-direct       v1, LTestDefaultPackage;-><init>()V
        10  (00000026) new-instance        v2, LTestDefaultPackage$TestInnerClass;
        11  (0000002a) invoke-virtual      v1, Ljava/lang/Object;->getClass()Ljava/
↪lang/Class;
        12  (00000030) invoke-direct       v2, v1, v6, v7, v8, LTestDefaultPackage
↪$TestInnerClass;-><init>(LTestDefaultPackage; I I LTestDefaultPackage
↪$TestInnerClass;)V
        13  (00000036) new-instance        v3, LTestDefaultPackage$TestInnerClass
↪$TestInnerInnerClass;
        14  (0000003a) invoke-virtual      v2, Ljava/lang/Object;->getClass()Ljava/
↪lang/Class;
        15  (00000040) invoke-direct       v3, v2, v6, v7, v8, LTestDefaultPackage
↪$TestInnerClass$TestInnerInnerClass;-><init>(LTestDefaultPackage$TestInnerClass; I␣
↪I LTestDefaultPackage$TestInnerClass$TestInnerInnerClass;)V
        16  (00000046) sget-object         v4, Ljava/lang/System;->out Ljava/io/
↪PrintStream;
        17  (0000004a) new-instance        v5, Ljava/lang/StringBuilder;
        18  (0000004e) const-string        v6, 't.a = '
        19  (00000052) invoke-direct       v5, v6, Ljava/lang/StringBuilder;-><init>
↪(Ljava/lang/String;)V
        20  (00000058) invoke-static       v2, LTestDefaultPackage$TestInnerClass;->
↪access$1(LTestDefaultPackage$TestInnerClass;)I
        21  (0000005e) move-result         v6
        22  (00000060) invoke-virtual      v5, v6, Ljava/lang/StringBuilder;->
↪append(I)Ljava/lang/StringBuilder;
        23  (00000066) move-result-object  v5
        24  (00000068) invoke-virtual      v5, Ljava/lang/StringBuilder;->
↪toString()Ljava/lang/String;
        25  (0000006e) move-result-object  v5
        26  (00000070) invoke-virtual      v4, v5, Ljava/io/PrintStream;->
↪println(Ljava/lang/String;)V
        27  (00000076) return-void
        28  (00000078) sget-object         v4, Ljava/lang/System;->out Ljava/io/
↪PrintStream;
        29  (0000007c) const-string        v5, '1 || 2'
        30  (00000080) invoke-virtual      v4, v5, Ljava/io/PrintStream;->
↪println(Ljava/lang/String;)V
        31  (00000086) goto                -53 [ main-BB@0x0000001c ]
        32  (00000088) sget-object         v4, Ljava/lang/System;->out Ljava/io/
↪PrintStream;
        33  (0000008c) const-string        v5, '3 || '
        34  (00000090) invoke-virtual      v4, v5, Ljava/io/PrintStream;->print(Ljava/
↪lang/String;)V
```

(continues on next page)

(continued from previous page)

```
        35 (00000096) goto                     -68 [ main-BB@0x0000000e ]
        36 (00000098) sget-object              v4, Ljava/lang/System;->out Ljava/io/
↪PrintStream;
        37 (0000009c) const-string             v5, '5'
        38 (000000a0) invoke-virtual           v4, v5, Ljava/io/PrintStream;->
↪println(Ljava/lang/String;)V
        39 (000000a6) goto                     -69 [ main-BB@0x0000001c ]
        40 (000000a8) packed-switch-payload
```

All these blocks are concatenated to each other. If you like, try to identify the basic blocks inside the dissassembly! Hint: The second column gives the offset inside the bytecode and matches the offset given in the CFG.

As you can see, the order of instructions in the bytecode does not match the execution order. For example, the *return* opcode is in the middle of the bytecode, while it is the end of the exection. Therefore some parts must have a *goto* to resume the execution at the correct point. For example, the basic block for the case that the argument of the switch opcode is `5` ends at offset 0xa6 and has a goto command to subtract 0x45 from the current offset. But that ends up beeing offset 0x61? No, it does not. To increase your confusion, you have to know, that offset arguments for opcodes are always in 16-bit units, while the offset used by androguard are counted in 8-bit units. That means, that you have to subtract 0x8a, which indeed returns to offset 0x1c in the bytecode.

> **Warning:** The offset units used are sometimes a little bit inconsistent across androguard! If you find some inconsistent behaviour, please report it as an issue.

To conclude, let's take a look at the actual Java source code of this particular method:

```java
public static void main(String [] z) {
    int a = 5;
    switch(a)
    {
    case 1:
    case 2:
        System.out.println("1 || 2");
        break;
    case 3:
        System.out.print("3 || ");
    case 4:
    default:
        System.out.println("4");
        break;
    case 5:
        System.out.println("5");
    }
    TestDefaultPackage p = new TestDefaultPackage();
    TestInnerClass t = p.new TestInnerClass(3, 4);
    TestInnerClass.TestInnerInnerClass t2 = t.new TestInnerInnerClass(3, 4);
    System.out.println("t.a = " + t.a);
}
```

Can you see how each Basic block belongs to a different path in the code?

1.1.5 Parsing Instructions and Bytecode

One often requested task is to parse the bytecode of all (or certain) methods. The bytecode can be used for various tasks, from creating simple statistics to machine learning.

The bytecode is stored for each method in the Dalvik file. Google provides some documentation about the bytecode format, which is very useful if you want to process it. Androguard can provide three different forms of the bytecode:

- Raw bytes
- disassembled representation
- decompiled representation

All three serve different purposes and might be used at the same time.

First of all, we need to know a few things about the differences of representation. While the documentation says, that bytecode is structured in 16bit units, Androguard will use 8bit units to show the bytecode (i.e. `bytes`). If offsets are given in the bytecode, they are also presented as bytes. Also all indices are provided in byte length. Other than that, the mnemonic representation should follow in large parts the one provided in the documentation. Arguments are always shown in their "expanded" form, which is especially important for the few opcodes where only parts of the value are stores, like `const/high16`. In this case, the full value is shown including the 16 lower zero bits. As Dalvik is closely related to Java, all integer values are represented as signed `int` (32bit value) or `long` (64bit). Values are either given in decimal or hexadecimal representation. If the value is hexadecimal, the value is suffixed with a `h`, i.e. `f7a0h` or `63392`.

In the following few examples, we will take the provided APK file `examples\android\TestsAndroguard\bin\TestActivi` `apk` and assume that you have loaded it via `AnalyzeAPK` and have the following objects:

```
>>> a
<androguard.core.bytecodes.apk.APK object at 0x00000000058DD240>
>>> d
[<androguard.core.bytecodes.dvm.DalvikVMFormat object at 0x0000000004CE4CF8>]
>>> dx
<analysis.Analysis VMs: 1, Classes: 492, Strings: 496>
```

Getting the raw bytecode

Our first task is to extract the raw bytecode of all methods.

```
for method in dx.get_methods():
    if method.is_external():
        continue
    # Need to get the EncodedMethod from the MethodClassAnalysis object
    m = method.get_method()
    if m.get_code():
        # get_code() returns None or a DalvikCode object
        # get_bc() returns a DCode object
        # get_raw() returns bytearray
        print(m.get_code().get_bc().get_raw())
```

This will print a lot of bytearrays.

Getting disassembled instructions

Next, we would like to get the disassembled instructions. The instruction itself have many different methods and you can find a detailed description in the documentation of the *`Instruction`* class.

```
for method in dx.get_methods():
    if method.is_external():
        continue
```

(continues on next page)

(continued from previous page)

```
    m = method.get_method()
    for idx, ins in m.get_instructions_idx():
        print(idx, ins.get_op_value(), ins.get_name(), ins.get_output())
```

This will print something like:

```
0 91 iput-object v1, v0, LTestDefaultPackage$TestInnerClass$TestInnerInnerClass;->this
↪$1 LTestDefaultPackage$TestInnerClass;
4 112 invoke-direct v0, Ljava/lang/Object;-><init>()V
10 89 iput v2, v0, LTestDefaultPackage$TestInnerClass$TestInnerInnerClass;->a I
14 89 iput v3, v0, LTestDefaultPackage$TestInnerClass$TestInnerInnerClass;->c I
18 14 return-void
```

The variable `idx` is the index counted in bytes where the opcode starts. `ins.get_op_value()` returns the integer value of the opcode, `ins.get_name()` the mnemonic and `ins.get_output()` the parsed arguments.

As an example, let's count the number of individual opcodes and create some statistics:

```
from collections import defaultdict
from operator import itemgetter
c = defaultdict(int)

for method in dx.get_methods():
    if method.is_external():
        continue
    m = method.get_method()
    for ins in m.get_instructions():
        c[(ins.get_op_value(), ins.get_name())] += 1

for k, v in sorted(c.items(), key=itemgetter(1), reverse=True)[:10]:
    print(k, '-->',  v)
```

This will output the top ten opcodes and the count:

```
(110, 'invoke-virtual') --> 3532
(84, 'iget-object') --> 2223
(12, 'move-result-object') --> 1749
(18, 'const/4') --> 1156
(112, 'invoke-direct') --> 1130
(10, 'move-result') --> 1111
(14, 'return-void') --> 1106
(56, 'if-eqz') --> 898
(26, 'const-string') --> 806
(113, 'invoke-static') --> 755
```

As another example, we will collect all constant integer values:

```
c = set()

for method in dx.get_methods():
    if method.is_external():
        continue
    m = method.get_method()
    for ins in m.get_instructions():
        if 0x12 <= ins.get_op_value() <= 0x19:
            c.add(ins.get_literals()[0])
```

(continues on next page)

(continued from previous page)

```
print('minimal:', min(c))
print('maximal:', max(c))
print('length: ', len(c))
```

This will print:

```
minimal: -4616189618054758400
maximal: 4707499256968118272
length:  205
```

Get processed bytecode from decompiler

The last topic is how to get the processed bytecode from the decompiler. If you are only interested in the decompiled source code, you can use the `source()` function:

```
for method in dx.get_methods():
    if method.is_external():
        continue
    m = method.get_method()
    print(m.source())
```

It will print all sources of all methods.

But, you can also use DAD to compile abstract syntax trees (AST) for you. An AST can easily be used to do analysis on the code itself. Unfortunately, the method to get to the AST is a little bit awkward:

```
from pprint import pprint
from androguard.decompiler.dad.decompile import DvMethod
for method in dx.get_methods():
    if method.is_external():
        continue
    dv = DvMethod(dx.get_method(method.get_method()))
    dv.process(doAST=True)
    pprint(dv.get_ast())
```

The AST is a dictionary, wich might look like this one:

```
{'body': ['BlockStatement',
          None,
          [['ExpressionStatement',
            ['Assignment',
             [['FieldAccess',
               [['Local', 'this']],
               (TestDefaultPackage$TestInnerClass$TestInnerInnerClass,
                this$1,
                LTestDefaultPackage$TestInnerClass;)],
              ['Local', 'p1']],
             '']],
           ['ExpressionStatement',
            ['Assignment',
             [['FieldAccess',
               [['Local', 'this']],
               (TestDefaultPackage$TestInnerClass$TestInnerInnerClass, a, I)],
              ['Local', 'p2']],
             '']],
```

(continues on next page)

(continued from previous page)

```
           ['ExpressionStatement',
            ['Assignment',
             [['FieldAccess',
               [['Local', 'this']],
               (TestDefaultPackage$TestInnerClass$TestInnerInnerClass, c, I)],
              ['Local', 'p3']],
             '']],
           ['ReturnStatement', None]]],
'comments': [],
 'flags': ['private'],
 'params': [[['TypeName', (TestDefaultPackage$TestInnerClass, 0)],
             ['Local', 'p1']],
            [['TypeName', ('.int', 0)], ['Local', 'p2']],
            [['TypeName', ('.int', 0)], ['Local', 'p3']]],
 'ret': ['TypeName', ('.void', 0)],
 'triple': (TestDefaultPackage$TestInnerClass$TestInnerInnerClass,
            <init>,
            (LTestDefaultPackage$TestInnerClass;II)V)}
```

This AST is the equivalent of the following source code:

```
private TestDefaultPackage$TestInnerClass$TestInnerInnerClass(TestDefaultPackage
↪$TestInnerClass p1, int p2, int p3)
{
    this.this$1 = p1;
    this.a = p2;
    this.c = p3;
    return;
}
```

1.1.6 Working with Sessions

If you are working on a larger APK, you might want to save your current work and come back later. Thats the reason for sessions: They allow you to save your work on disk and resume it at any point. Sessions could also be used to store the analysis on disk, for example if you do automated analysis and want to analyse certain files later.

There are several ways to work with sessions. The easiest way is to use `AnalyzeAPK()` with a session:

```
from androguard import misc
from androguard import session

# get a default session
sess = misc.get_default_session()

# Use the session
a, d, dx = misc.AnalyzeAPK("examples/android/abcore/app-prod-debug.apk", session=sess)

# Show the current Session information
sess.show()

# Do stuff...

# Save the session to disk
session.Save(sess, "androguard_session.ag")

```

(continues on next page)

(continued from previous page)

```
# Load it again
sess = session.Load("androguard_session.ag")
```

The session information will look like this:

```
APKs in Session: 1
    d5e26acca809e9cdfaece18afd8e63c60a26d7b6d566d70bd9f44d6934d5c433: [<androguard.
↪core.bytecodes.apk.APK object at 0x7fcecf4f3f10>]
DEXs in Session: 2
    8bd7e9f48a6ed29e4c678633364e8bfd4e6ae76ef3e50c43a5ec3c00eb10a5bc: <analysis.
↪Analysis VMs: 2, Classes: 3092, Strings: 3293>
    e2a1e46ecd03b701ce72c31057581e0104279d142fca06cdcdd000dd94a459e0: <analysis.
↪Analysis VMs: 2, Classes: 3092, Strings: 3293>
Analysis in Session: 1
    d5e26acca809e9cdfaece18afd8e63c60a26d7b6d566d70bd9f44d6934d5c433: <analysis.
↪Analysis VMs: 2, Classes: 3092, Strings: 3293>
```

Similar functionality is available from the Session directly, but needs a second function to retrive the analyzed objects from the Session:

```
from androguard.session import Session

s = Session()
sha256 = s.add("examples/android/abcore/app-prod-debug.apk")

a, d, dx = s.get_objects_apk(digest=sha256)

s.show()

# When no filename is given, the Session will be saved at the current directory
saved_file = s.save()
# ... and return the filename of the Session file
print(saved_file)
```

Note: Session objects store a lot of data and can get very big!

It is recommended not to use sessions in automated environments, where hundrets or thousands of APKs are loaded.

If you want to use sessions but keep the session alive only for one or multiple APKs, you can call the *reset()* method on a session, to remove all stored analysis data.

```
from androguard import misc
from androguard import session
import os

# get a default session
sess = misc.get_default_session()

for root, dirs, files in os.walk("examples")
    for f in files:
        if f.endswith(".apk"):
            # Use the session
            a, d, dx = misc.AnalyzeAPK(os.path.join(root, f), session=sess)

            # Do your stuff
```

(continues on next page)

(continued from previous page)

```
            # Maybe save the session to disk...

            # But now reset the session for the next analysis
            sess.reset()
```

1.1.7 Use JADX as a Decompiler

Instead of using the internal decompiler DAD, you can also use JADX.

Install JADX as described at it's website. Make sure that the `jadx` executable is in `$PATH`. Otherwise you might set the argument when calling `DecompilerJADX()`.

Here is a short demo code, how JADX can be used:

```
from androguard.core.bytecodes.apk import APK
from androguard.core.bytecodes.dvm import DalvikVMFormat
from androguard.core.analysis.analysis import Analysis
from androguard.decompiler.decompiler import DecompilerJADX
from androguard.core.androconf import show_logging
import logging

# Enable log output
show_logging(level=logging.DEBUG)

# Load our example APK
a = APK("examples/android/TestsAndroguard/bin/TestActivity.apk")

# Create DalvikVMFormat Object
d = DalvikVMFormat(a)
# Create Analysis Object
dx = Analysis(d)

# Load the decompiler
# Make sure that the jadx executable is found in $PATH
# or use the argument jadx="/path/to/jadx" to point to the executable
decompiler = DecompilerJADX(d, dx)

# propagate decompiler and analysis back to DalvikVMFormat
d.set_decompiler(decompiler)
d.set_vmanalysis(dx)

# Now you can do stuff like:
for m in d.get_methods()[:10]:
    print(m)
    print(decompiler.get_source_method(m))
```

1.1.8 Android Signing Certificates

Androguard has the ability to get information about the signing certificate found in APKs. Over the last versions of Androguard, different parsers has been used to get certificate information. The first parser was Chilkat, then a mixture of pyasn1 and cryptography was used, while the latest parser uses the asn1crypto library. Not all x509 parsers work with all certificates as there are plenty of examples where the certificate creator does not follow the RFCs for creating certificates. Some parsers do not accept such broken certificates and will fail to parse them.

The purpose of Androids signing process is not to provide verified information about the author, like with JAR signing, but only provide a way to check the integrity of the APK as well as check if an APK can be upgraded by comparing the certificate fingerprints. In some sense, the certificate information can be used to find other APKs from the same author - as long as the signing key was kept secret! There are also public available signing keys, like the ones from AOSP, thus the same fingerprint of two APKs does not always tell you it was signed by the same person.

If you like to know more about the APK signing process, please read the official documentation about Signing. There is also an official tool to verify and sign APKs called apksigner.

Working with certificates

Inside the APK, there are two places for certificates:

- v1 aka JAR signing: PKCS#7 files in the `META-INF` folder
- v2 aka APK signing: a special section in the ZIP containing DER coded certifcates

The easiest way to get to the certificate information is *androguard sign - Print Certificate Fingerprints*. It gives similar output to apksigner, but uses only androguard. It can not verify the integrity of the file though.

```
$ androsign.py --all --show examples/signing/apksig/golden-aligned-v1v2-out.apk
golden-aligned-v1v2-out.apk, package: 'android.appsecurity.cts.tinyapp'
Is signed v1: True
Is signed v2: True
Found 1 unique certificates
Issuer: CN=rsa-2048
Subject: CN=rsa-2048
Serial Number: 0x8e35306cdd0115f7L
Hash Algorithm: sha256
Signature Algorithm: rsassa_pkcs1v15
Valid not before: 2016-03-31 14:57:49+00:00
Valid not after: 2043-08-17 14:57:49+00:00
sha1 0aa07c0f297b4ae834dc85a17eea8c2cf9380ff7
sha256 fb5dbd3c669af9fc236c6991e6387b7f11ff0590997f22d0f5c74ff40e04fca8
sha512␣
↪4da6e6744a4dabef192b198be13b4492b0ce97469f3ce223dd9b7e8df2ee952328e06651e5e65dd3b60ac5e3946e16cf705
md5 e995a5ed7137307661f854e66901ee9e
```

As a comparison, here is the output of apksigner:

```
$ apksigner verify -verbose --print-certs examples/signing/apksig/golden-aligned-v1v2-
↪out.apk
Verifies
Verified using v1 scheme (JAR signing): true
Verified using v2 scheme (APK Signature Scheme v2): true
Number of signers: 1
Signer #1 certificate DN: CN=rsa-2048
Signer #1 certificate SHA-256 digest:␣
↪fb5dbd3c669af9fc236c6991e6387b7f11ff0590997f22d0f5c74ff40e04fca8
Signer #1 certificate SHA-1 digest: 0aa07c0f297b4ae834dc85a17eea8c2cf9380ff7
Signer #1 certificate MD5 digest: e995a5ed7137307661f854e66901ee9e
Signer #1 key algorithm: RSA
Signer #1 key size (bits): 2048
Signer #1 public key SHA-256 digest:␣
↪8cabaedf32f1052f6bc5edbeb84d1c500f8c1aa15f8944bf22c46e44c5c4f7e8
Signer #1 public key SHA-1 digest: a708f9a777bac814e6634b02521224537ec3e019
Signer #1 public key MD5 digest: c0c8801fabf2ad970282be1c41584003
```

The most interesting part is probaby the fingerprint of the certificate (not of the public key!). You can use it to search for similar APKs. Sometimes there is a confusion about this fingerprint: The fingerprint is not the checksum of the whole PKCS#7 file, but only of a certain part of it! Calculating the hash of a PKCS#7 file from two different, but equally signed APKs will result in a different hash. The fingerprint will stay the same though.

Androguard offers methods in the *androguard.core.bytecodes.apk.APK* class to iterate over the certificates found there.

```
from androguard.core.bytecodes.apk import APK

a = APK('examples/signing/apksig/golden-aligned-v1v2-out.apk')

# first check if this APK is signed
print("APK is signed: {}".format(a.is_signed()))

if a.is_signed():
    # Test if signed v1 or v2 or both
    print("APK is signed with: {}".format("both" if a.is_signed_v1() and
    a.is_signed_v2() else "v1" if a.is_signed_v1() else "v2"))

# Iterate over all certificates
for cert in a.get_certificates():
    # Each cert is now a asn1crypt.x509.Certificate object
    # From the Certificate object, we can query stuff like:
    cert.sha1  # the sha1 fingerprint
    cert.sha256  # the sha256 fingerprint
    cert.issuer.human_friendly  # issuer
    cert.subject.human_friendly  # subject, usually the same
    cert.hash_algo  # hash algorithm
    cert.signature_algo  # Signature algorithm
    cert.serial_number  # Serial number
    cert.contents  # The DER coded bytes of the certificate itself
    # ...
```

Please referr to the asn1crypto documentation for more information on the features of the `Certificate` class!

1.1.9 Android Binary XML Format

Android uses a special format to save XML and resource files. Also resource files are XML files in the source folder, but all resources are packed into a single resource file called `resources.arsc`. The underlying format is chunk based and is capable for storing several different information.

The most common AXML file is the `AndroidManifest.xml`. This file must be part of every APK, and contains the meta-information about the package.

Androguard is capable of decoding such files and two different tools exists for decoding:

1) `androguard arsc` for decoding `resources.arsc`.
2) `androguard axml` for decoding `AndroidManifest.xml` and all other XML files

Decode the AndroidManifest.xml

Let's use one of the example files provided by androguard. To decode the AndroidManifest.xml of an APK file, simply give `androguard axml` the APK file as an argument:

```
$ androguard axml examples/android/TestsAndroguard/bin/TestActivity.apk
```

The output will look like this:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"␣
↪android:versionCode="1" android:versionName="1.0" package="tests.androguard">
  <uses-sdk android:minSdkVersion="9" android:targetSdkVersion="16"/>
  <application android:label="@7F040001" android:icon="@7F020000" android:debuggable=
↪"true" android:allowBackup="false">
    <activity android:label="@7F040001" android:name="TestActivity">
      <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER"/>
      </intent-filter>
    </activity>
  </application>
</manifest>
```

You can check with the original, uncompiled, XML file, which can be found here:

```
$ cat examples/android/TestsAndroguard/AndroidManifest.xml
```

The original file will print:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="tests.androguard"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="9"
        android:targetSdkVersion="16" />

    <application
        android:allowBackup="false"
        android:icon="@drawable/icon"
        android:label="@string/app_name" >
        <activity
            android:name="TestActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
```

Note, that the overall structure is equal but there are certain differences.

1) Resource labels are hex numbers in the decompiled version but strings in the original one

2) Newlines and whitespaces are different.

Due to the compilation, this information is lost. But it does not matter, as the structure of the Manifest does not matter. To get some information about the resource IDs, we need information from the `resources.arsc`.

To retrive information about a single ID, simply run the following:

```
$ androguard arsc examples/android/TestsAndroguard/bin/TestActivity.apk  --id 7F040001
@7f040001 resolves to '@tests.androguard:string/app_name'

<default> = 'TestsAndroguardApplication'
```

You can see, that the ID `7F040001` was successfully resolved to the same string from the source file. To understand how Android handles resource configurations, you should read HandlingResources.

Decode any other XML file

Also layout files or other XML files provided with the APK are compiled. To decompile them, just give the path inside the APK as an argument, or specify the binary XML file directly:

```
$ androguard axml examples/android/TestsAndroguard/bin/TestActivity.apk -r res/layout/
↪main.xml
$ androguard axml examples/axml/test.xml
```

Decode information from the resources.arsc

To get XML resource files out of the binary `resources.arsc`, use `androguard arsc`.

For example, get all string resources of an APK:

```
$ androguard arsc examples/android/TestsAndroguard/bin/TestActivity.apk --type string
```

will give the following output:

```
<resources>
<string name="hello">Hello World, TestActivity! kikoololmodif</string>
<string name="app_name">TestsAndroguardApplication</string>
</resources>
```

You can also list all resource types:

```
$ androguard arsc examples/android/TestsAndroguard/bin/TestActivity.apk --list-types
In Package: tests.androguard
  In Locale: \x00\x00
    drawable
    layout
    public
    string
```

Working with AXML and Resource files from python

To load an AXML file, for example the `AndroidManifest.xml`, use the *`AXMLPrinter`*:

```
from androguard.core.bytecodes.axml import AXMLPrinter
with open("AndroidManifest.xml", "rb") as fp:
    a = AXMLPrinter(fp.read())

# Get the lxml.etree.Element from the AXMLPrinter:
xml = a.get_xml_obj()

```

(continues on next page)

(continued from previous page)

```
# For example, get all uses-permission:
xml.findall("uses-permission")
```

In order to use resources, you need the `ARSCParser`:

```
from androguard.core.bytecodes.axml import ARSCParser

with open("resouces.arsc", "rb") as fp:
    res = ARSCParser(fp.read())

# Now you can resolve IDs:
name = res.get_resource_xml_name(0x7F040001)
if name:
    print(name)

# To get the content of an ID, you need to iterate over configurations
# You need to decide which configuration to use...
for config, entry in res.get_res_configs(0x7F040001):
    # You can query `config` for specific configuration
    # or check with `is_default()` if this is a default configuration.
    print("{} = '{}'".format(config.get_qualifier() if not config.is_default() else "
↪<default>", entry.get_key_data()))
```

1.1.10 Bulk Analysis

Androguard is capable of analysing probably thousand to millions of APKs. It is also possible to use tools like *multiprocessing* for this job and analyse APKs in parallel. Usually you want to put the results of your analysis somewhere, for example a database or some log file. It is also possile to use `Session` objects for this job, but you should be aware of some caveats!

1) Sessions take up a lot of space per APK. The resulting Session object can be more than 30 times larger than the original APK 2) Sessions should not be used to add unrelated APKs, again the size will blow up and you need to figure out which APK belongs to where

So the rule of thumb would be to not use Sessions for bulk analysis, only if you know what you are doing. Another way is to pickle the resulting objects. As the `DalvikVMFormat` are already stored in the `Analysis` object, there is no need to pickle them separately. Thus, it is only required to save the `APK` and `Analysis` object.

This is an example how to obtain the two objects and saving them to disk:

```
import sys
from pickle import dump
from hashlib import sha512
from androguard.misc import AnalyzeAPK

a, _, dx = AnalyzeAPK('examples/tests/a2dp.Vol_137.apk')

sha = sha512()

sha.update(a.get_raw())

with open("{}_apk.p".format(sha.hexdigest()), "wb") as fp:
    dump(a, fp)

with open("{}_analysis.p".format(sha.hexdigest()), "wb") as fp:
```

(continues on next page)

(continued from previous page)

```
    # It looks like here is the recursion problem...
    sys.setrecursionlimit(50000)
    dump(dx, fp)
```

But the resulting files are very large, especially the Analysis package:

```
$ du -sh examples/tests/a2dp.Vol_137.apk
808K examples/tests/a2dp.Vol_137.apk

$ du -sh *.p
31M ␣
↪24a62690a770891a8f43d71e8f7beb24821d46a75e017ef4f4e6a04624105466621c96305d8e86f9900042e3ef1d5806a5c
↪analysis.p
852K␣
↪24a62690a770891a8f43d71e8f7beb24821d46a75e017ef4f4e6a04624105466621c96305d8e86f9900042e3ef1d5806a5c
↪apk.p
```

But it is possible to compress both files to save disk space:

```
import sys
import lzma
from pickle import dump
from hashlib import sha512
from androguard.misc import AnalyzeAPK

a, _, dx = AnalyzeAPK('examples/tests/a2dp.Vol_137.apk')

sha = sha512()

sha.update(a.get_raw())

with lzma.open("{}_apk.p.lzma".format(sha.hexdigest()), "wb") as fp:
    dump(a, fp)

with lzma.open("{}_analysis.p.lzma".format(sha.hexdigest()), "wb") as fp:
    # It looks like here is the recursion problem...
    sys.setrecursionlimit(50000)
    dump(dx, fp)
```

which results in much smaller files:

```
$ du -sh *.lzma
4,5M␣
↪24a62690a770891a8f43d71e8f7beb24821d46a75e017ef4f4e6a04624105466621c96305d8e86f9900042e3ef1d5806a5c
↪analysis.p.lzma
748K␣
↪24a62690a770891a8f43d71e8f7beb24821d46a75e017ef4f4e6a04624105466621c96305d8e86f9900042e3ef1d5806a5c
↪apk.p.lzma
```

Obviously, as the APK is already packed, there is not much to compress anymore.

Using AndroAuto

Another method is to use the framework *AndroAuto*. AndroAuto allows you to write small python classes which implement some method, which are then called by AndroAuto at certain points in time. AndroAuto is capable of analysing thousands of apps, and uses threading to distribute the load to multiple CPUs. The results of your analysis can then be dumped to disk, or you could write your own method of saving them - for example, in a database.

The two key components are a Logger, for example `DefaultAndroLog` and an Analysis Runner, for example `DefaultAndroAnalysis`. Both are passed via a settings dictionary into `AndroAuto`.

Next, a minimal working example is given:

```
from androguard.core.analysis import auto
import sys

class AndroTest(auto.DirectoryAndroAnalysis):
    def __init__(self, path):
       super(AndroTest, self).__init__(path)
       self.has_crashed = False

    def analysis_app(self, log, apkobj, dexobj, analysisobj):
        # Just print all objects to stdout
        print(log.id_file, log.filename, apkobj, dexobj, analysisobj)

    def finish(self, log):
       # This method can be used to save information in `log`
       # finish is called regardless of a crash, so maybe store the
       # information somewhere
       if self.has_crashed:
          print("Analysis of {} has finished with Errors".format(log))
       else:
          print("Analysis of {} has finished!".format(log))

    def crash(self, log, why):
       # If some error happens during the analysis, this method will be
       # called
       self.has_crashed = True
       print("Error during analysis of {}: {}".format(log, why), file=sys.stderr)

settings = {
    # The directory `some/directory` should contain some APK files
    "my": AndroTest('some/directory'),
    # Use the default Logger
    "log": auto.DefaultAndroLog,
    # Use maximum of 2 threads
    "max_fetcher": 2,
}

aa = auto.AndroAuto(settings)
aa.go()
```

In this example, the `analysis_app()` function is used to get all created objects of the analysis and just print them to stdout.

More information can be found in the documentation of `AndroAuto`.

1.1.11 Debugging Broken APKs

Sometimes you will have troubles to get something done with androguard. This is usually the case if an APK uses some edge cases or deliberately tries to break parsers - which is not uncommon for malware.

Please feel free to open a bug report in such cases, so this error can be fixed. But before you do, try to gather some more information about the APK. Sometimes not only androguard failes to decode the file, but the official tools do as well!

It is also always interesting to know, if such a broken file can still be installed on an Android system. If you like to test this, fire up an emulator and try to run the APK there.

AXML Parser / AndroidManifest.xml

Many errors happen in the parsing of the *AndroidManifest.xml*.

There are two official tools you can use to decode the *AndroidManifest.xml*:

1. aapt2
2. apkanalyzer

Both are available in the AndroidSDK. While aapt2 can only decode the structure of the file, apkanalyzer can give an actual XML:

```
$ apkanalyzer manifest print org.fdroid.fdroid_1002052.apk | head
<?xml version="1.0" encoding="utf-8"?>
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:versionCode="1002052"
    android:versionName="1.2.2"
    android:installLocation="0"
    package="org.fdroid.fdroid"
    platformBuildVersionCode="24"
    platformBuildVersionName="7.0">

$ aapt2 dump org.fdroid.fdroid_1002052.apk --file AndroidManifest.xml | head
Binary XML
N: android=http://schemas.android.com/apk/res/android (line=2)
  E: manifest (line=2)
    A: http://schemas.android.com/apk/res/android:versionCode(0x0101021b)=1002052
    A: http://schemas.android.com/apk/res/android:versionName(0x0101021c)="1.2.2"␣
↪(Raw: "1.2.2")
    A: http://schemas.android.com/apk/res/android:installLocation(0x010102b7)=0
    A: package="org.fdroid.fdroid" (Raw: "org.fdroid.fdroid")
    A: platformBuildVersionCode=24 (Raw: "24")
    A: platformBuildVersionName=7 (Raw: "7.0")
      E: uses-sdk (line=8)
```

Both outputs are actually useful, as aapt2 can provide much more detailed information about the format than apkanalyzer does.

Broken ZIP files

As you might know, APK files are actually just ZIP files. You can test the zip file integrity using the ZIP command itself:

```
$ zip -T org.fdroid.fdroid_1002052.apk
test of org.fdroid.fdroid_1002052.apk OK
```

If there are any errors, like wrong CRC32, these get reported here. Other ZIP implementations have similar tools to check ZIP files.

Verifying the APK Signature

You can check the signature of the file using apksigner from the AndroidSDK:

```
$ apksigner verify --verbose --print-certs org.fdroid.fdroid_1002052.apk
Verifies
Verified using v1 scheme (JAR signing): true
Verified using v2 scheme (APK Signature Scheme v2): false
Number of signers: 1
Signer #1 certificate DN: CN=Ciaran Gultnieks, OU=Unknown, O=Unknown, L=Wetherby,␣
↪ST=Unknown, C=UK
Signer #1 certificate SHA-256 digest:␣
↪43238d512c1e5eb2d6569f4a3afbf5523418b82e0a3ed1552770abb9a9c9ccab
Signer #1 certificate SHA-1 digest: 05f2e65928088981b317fc9a6dbfe04b0fa13b4e
Signer #1 certificate MD5 digest: 17c55c628056e193e95644e989792786
Signer #1 key algorithm: RSA
Signer #1 key size (bits): 2048
Signer #1 public key SHA-256 digest:␣
↪e3d2cc87a245da2e84d4fb71e527c164e084d48bccf76ffad46ad17f1bfde388
Signer #1 public key SHA-1 digest: 26ef7882633282a9b04688178ee7f372fbec7c3d
Signer #1 public key MD5 digest: 9225fccafb33b605a86cfc09d7f38ec6
WARNING: META-INF/rxandroid.properties not protected by signature. Unauthorized␣
↪modifications to this JAR entry will not be detected. Delete or move the entry␣
↪outside of META-INF/.
WARNING: META-INF/rxjava.properties not protected by signature. Unauthorized␣
↪modifications to this JAR entry will not be detected. Delete or move the entry␣
↪outside of META-INF/.
WARNING: META-INF/services/com.fasterxml.jackson.core.JsonFactory not protected by␣
↪signature. Unauthorized modifications to this JAR entry will not be detected.␣
↪Delete or move the entry outside of META-INF/.
WARNING: META-INF/services/com.fasterxml.jackson.core.ObjectCodec not protected by␣
↪signature. Unauthorized modifications to this JAR entry will not be detected.␣
↪Delete or move the entry outside of META-INF/.
WARNING: META-INF/buildserverid not protected by signature. Unauthorized␣
↪modifications to this JAR entry will not be detected. Delete or move the entry␣
↪outside of META-INF/.
WARNING: META-INF/fdroidserverid not protected by signature. Unauthorized␣
↪modifications to this JAR entry will not be detected. Delete or move the entry␣
↪outside of META-INF/.
```

1.2 Tools

The only tool you need is *androguard - The swiss army knife*. It combines all old tools into a single command line interface.

You can still use the other tools as well, but note that they might get removed some day.

1.2.1 androguard - The swiss army knife

androguard is the new tool, which combines all the other tools into a single command line interface application.

```
Usage: androguard [OPTIONS] COMMAND [ARGS]...

  Androguard is a full Python tool to play with Android files.
```

(continues on next page)

(continued from previous page)

```

Options:
  --version           Show the version and exit.
  --verbose, --debug  Print more
  --quiet             Print less (only warnings and above)
  --silent            Print no log messages
  --help              Show this message and exit.

Commands:
  analyze      Open a IPython Shell and start reverse engineering.
  apkid        Return the packageName/versionCode/versionName per APK as...
  arsc         Decode resources.arsc either directly from a given file or...
  axml         Parse the AndroidManifest.xml.
  cg           Create a call graph and export it into a graph format.
  decompile    Decompile an APK and create Control Flow Graphs.
  disassemble  Disassemble Dalvik Code with size SIZE starting from an...
  gui          Androguard GUI
  sign         Return the fingerprint(s) of all certificates inside an APK.
```

Take a look at the detailed description of each tool in the next sections.

1.2.2 androguard analyze - Androguard Shell

androlyze is a tool that spawns an IPython shell.

```
Usage: androguard analyze [OPTIONS] [APK]

  Open a IPython Shell and start reverse engineering.

Options:
  --session PATH  Previously saved session to load instead of a file
  --help          Show this message and exit.
```

1.2.3 androguard cg - Create Call Graph from APK

androcg can create files that can be read using graph visualization software, for example gephi.

Synopsis

```
Usage: androguard cg [OPTIONS] APK

  Create a call graph and export it into a graph format.

  The default is to create a file called callgraph.gml in the current
  directory!

  classnames are found in the type "Lfoo/bar/bla;".

  Example:

      $ androguard cg examples/tests/hello-world.apk

```

(continues on next page)

(continued from previous page)

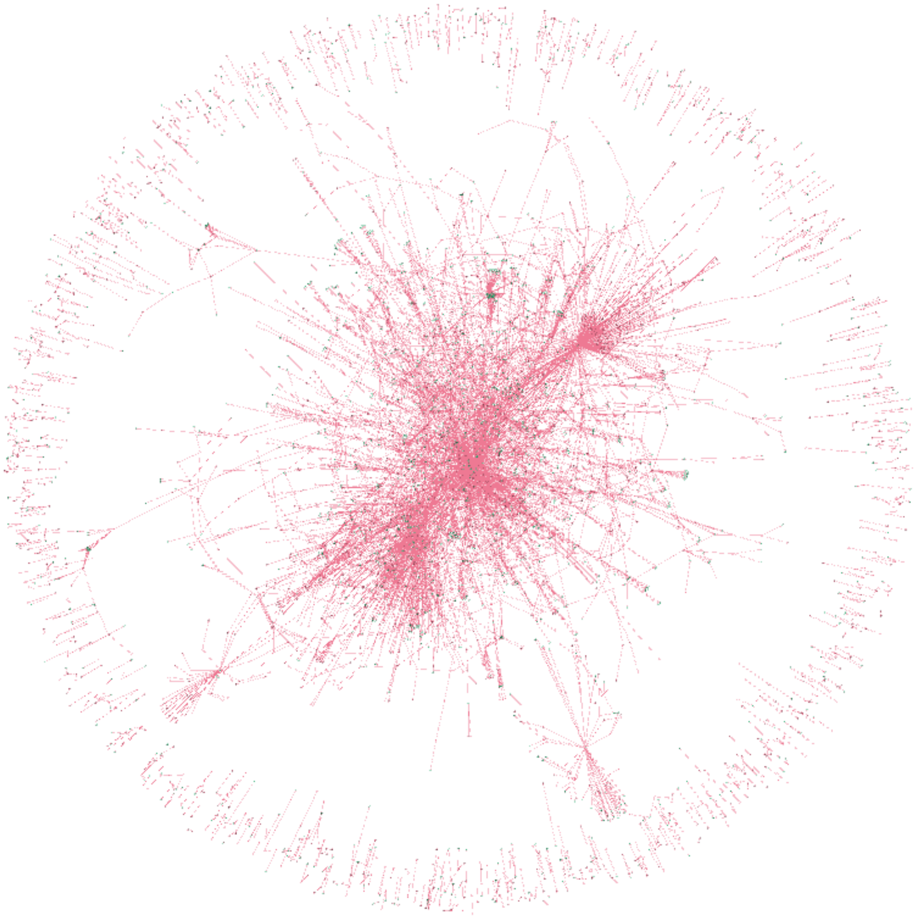
```
Options:
  -o, --output TEXT           Filename of the output file, the extension is
                              used to decide which format to use  [default:
                              callgraph.gml]
  -s, --show                  instead of saving the graph, print it with
                              mathplotlib (you might not see anything!)
  -v, --verbose               Print more output
  --classname TEXT            Regex to filter by classname  [default: .*]
  --methodname TEXT           Regex to filter by methodname  [default: .*]
  --descriptor TEXT           Regex to filter by descriptor  [default: .*]
  --accessflag TEXT           Regex to filter by accessflags  [default: .*]
  --no-isolated / --isolated  Do not store methods which has no xrefs
  --help                      Show this message and exit.
```

Examples

The call graph is constructed from the `Analysis` object and then converted into a networkx *MultiDiGraph*. Currently supported formats are gml, gexf, gpickle, graphml, yaml and net.

The call graph contains methods as nodes and calls as edges. Each edge has the offset inside the method stored as an attribute and multiple calls between two methods result in multiple edges.

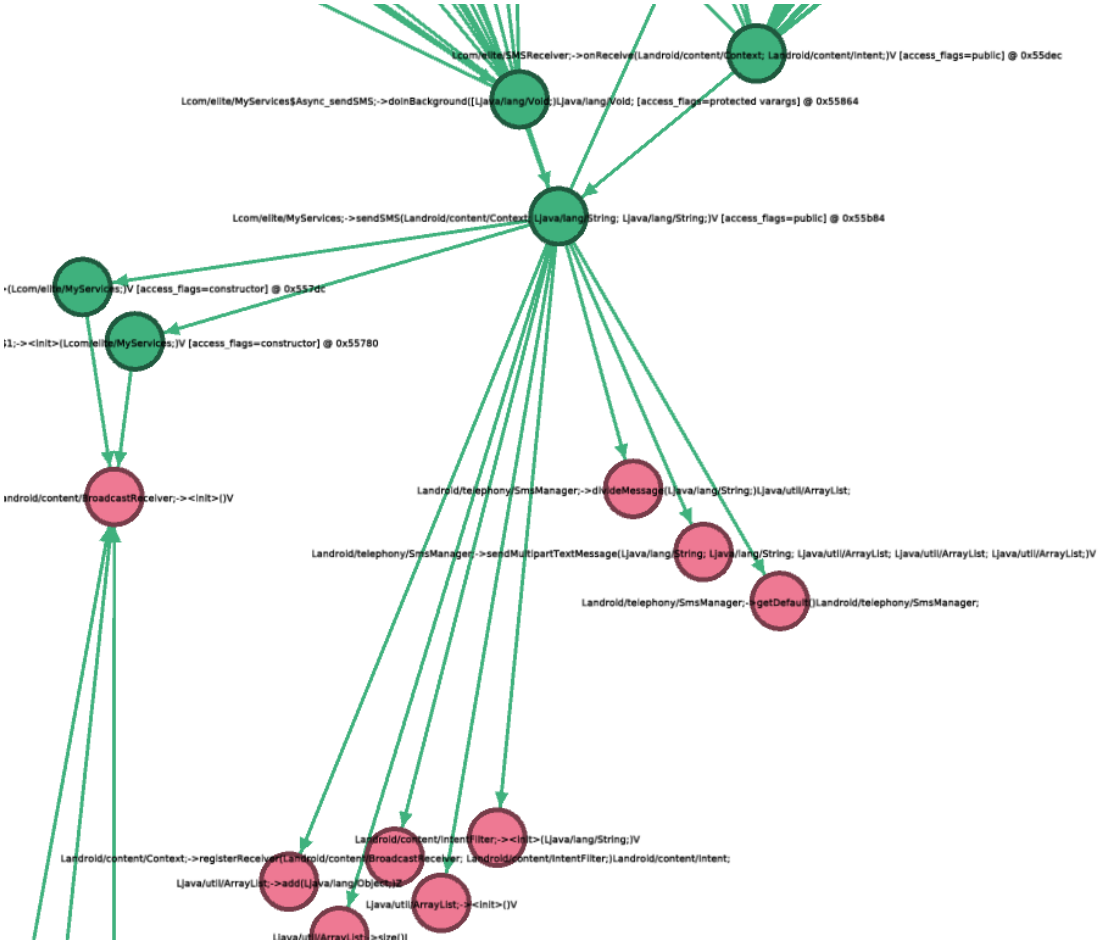
The methods to construct the callgraph from can be filtered. It is highly suggested to do that, as call graphs can get very large:



Of course, you can export the call graph with androguard and filter it later. If you filter with androguard, calls to this method will be stored, as well as calls from the method. For external methods only the first direction can be saved.

Note: External methods are not automatically API methods! It might be the case, that the method in question could not be found during disassembly time, hence it is stored as external.

Here is an example of an already filtered graph, visualized in gephi. Each node has an attribute to indicate if it is an internal (defined somewhere in the DEXs) or external (might be an API, but definetly not defined in the DEXs) method. In this case all green nodes are internal and all red ones are external. You can see the calls of some SMS Trojan to the API methods to write SMS.


Lcom/elite/SMSReceiver;->onReceive(Landroid/content/Context; Landroid/content/Intent;)V [access_flags=public] @ 0x55dec
Lcom/elite/MyServices$Async_sendSMS;->doInBackground([Ljava/lang/Void;)Ljava/lang/Void; [access_flags=protected varargs] @ 0x55864
Lcom/elite/MyServices;->sendSMS(Landroid/content/Context; Ljava/lang/String; Ljava/lang/String;)V [access_flags=public] @ 0x55b84
(Lcom/elite/MyServices;)V [access_flags=constructor] @ 0x557dc
$1;-><init>(Lcom/elite/MyServices;)V [access_flags=constructor] @ 0x55780
ndroid/content/BroadcastReceiver;-><init>()V
Landroid/telephony/SmsManager;->divideMessage(Ljava/lang/String;)Ljava/util/ArrayList;
Landroid/telephony/SmsManager;->sendMultipartTextMessage(Ljava/lang/String; Ljava/lang/String; Ljava/util/ArrayList; Ljava/util/ArrayList; Ljava/util/ArrayList;)V
Landroid/telephony/SmsManager;->getDefault()Landroid/telephony/SmsManager;
Landroid/content/IntentFilter;-><init>(Ljava/lang/String;)V
Landroid/content/Context;->registerReceiver(Landroid/content/BroadcastReceiver; Landroid/content/IntentFilter;)Landroid/content/Intent;
Ljava/util/ArrayList;->add(Ljava/lang/Object;)Z
Ljava/util/ArrayList;-><init>()V
Ljava/util/ArrayList;->size()I

1.2.4 androguard gui - Androguard GUI

Warning: The androgui is experimental and might not fully work!

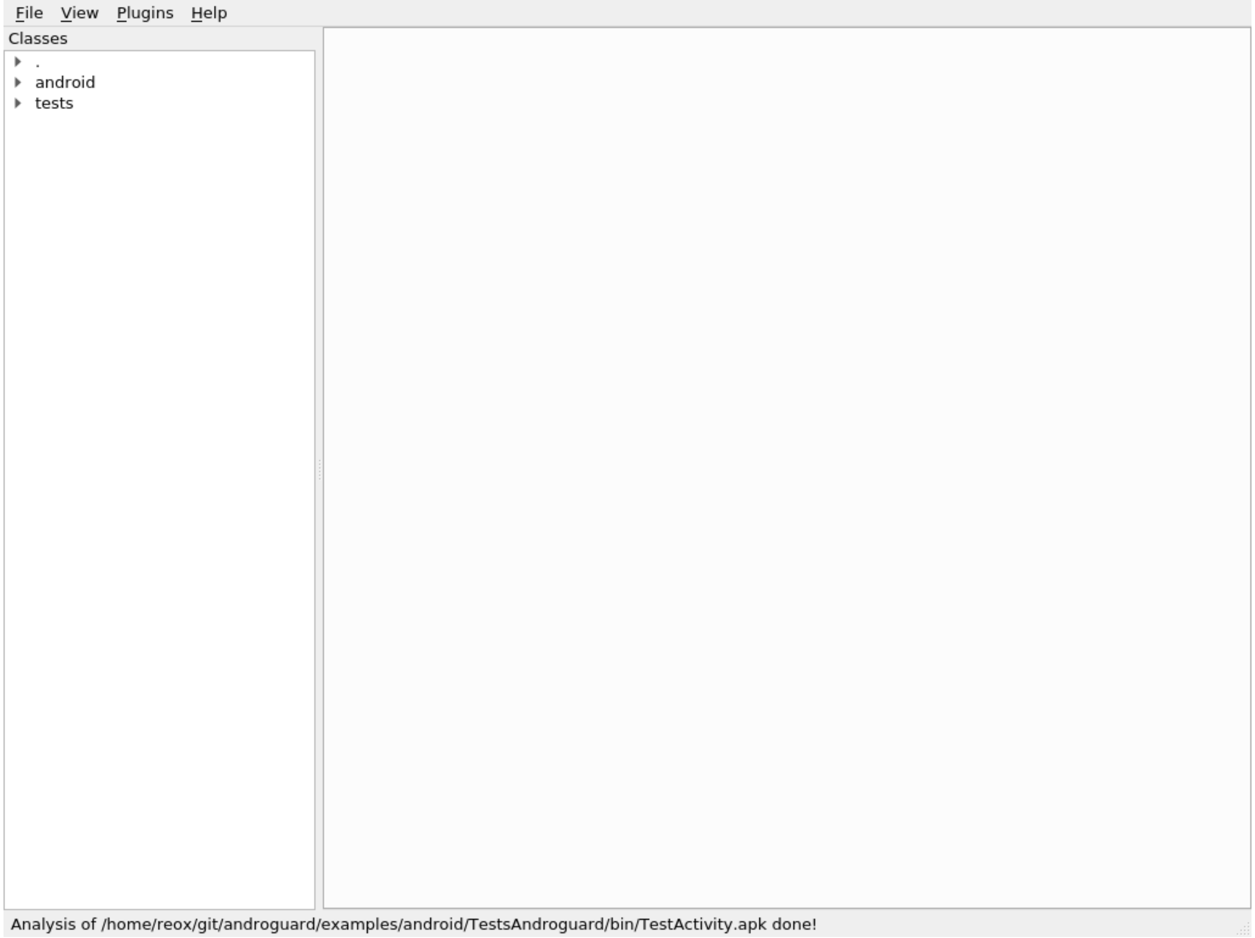
```
Usage: androguard gui [OPTIONS]

  Androguard GUI

Options:
  -i, --input_file FILE    Specify the inital file to load in the GUI
  -p, --input_plugin PATH  Additional Plugin (currently unused)
  --help                   Show this message and exit.
```

Examples

The androguard gui currently has functions to show disassmebled dalvik code, print all strings, methods, API usage and resources.

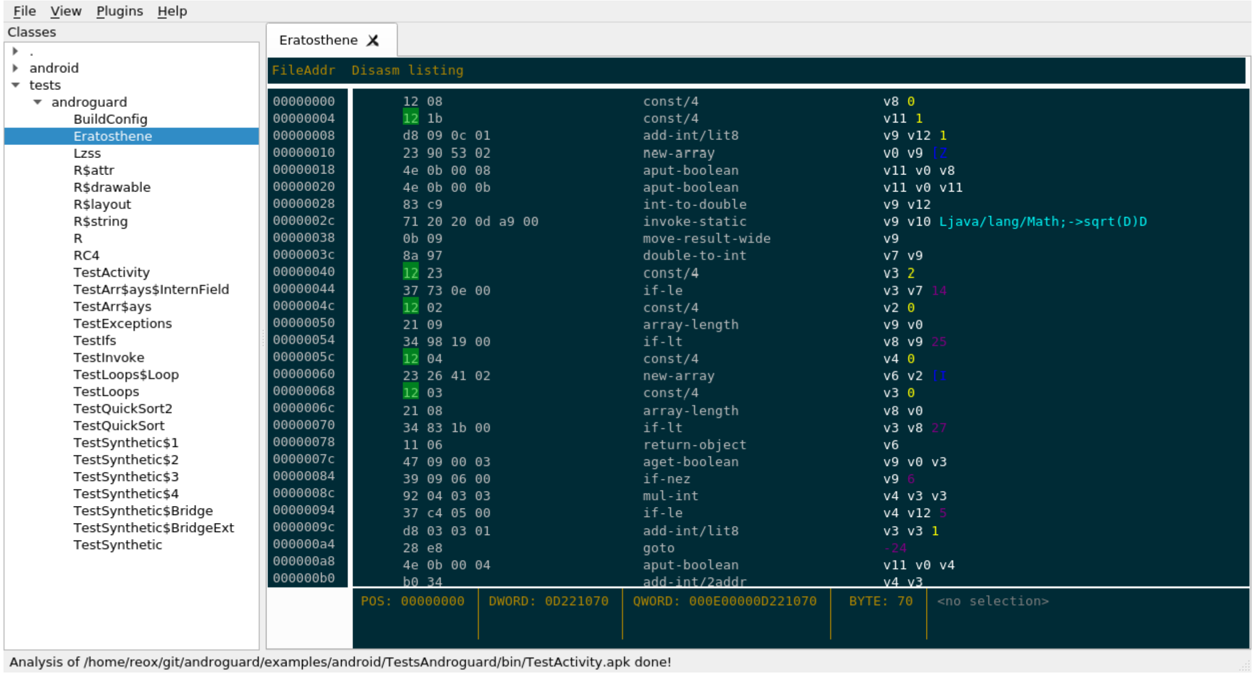
It uses `Session` in order to resume the work later.

First, open up an APK using File, Open. If everything has worked, you will see all classes found inside the APK in the left tree view:



Analysis of /home/reox/git/androguard/examples/android/TestsAndroguard/bin/TestActivity.apk done!

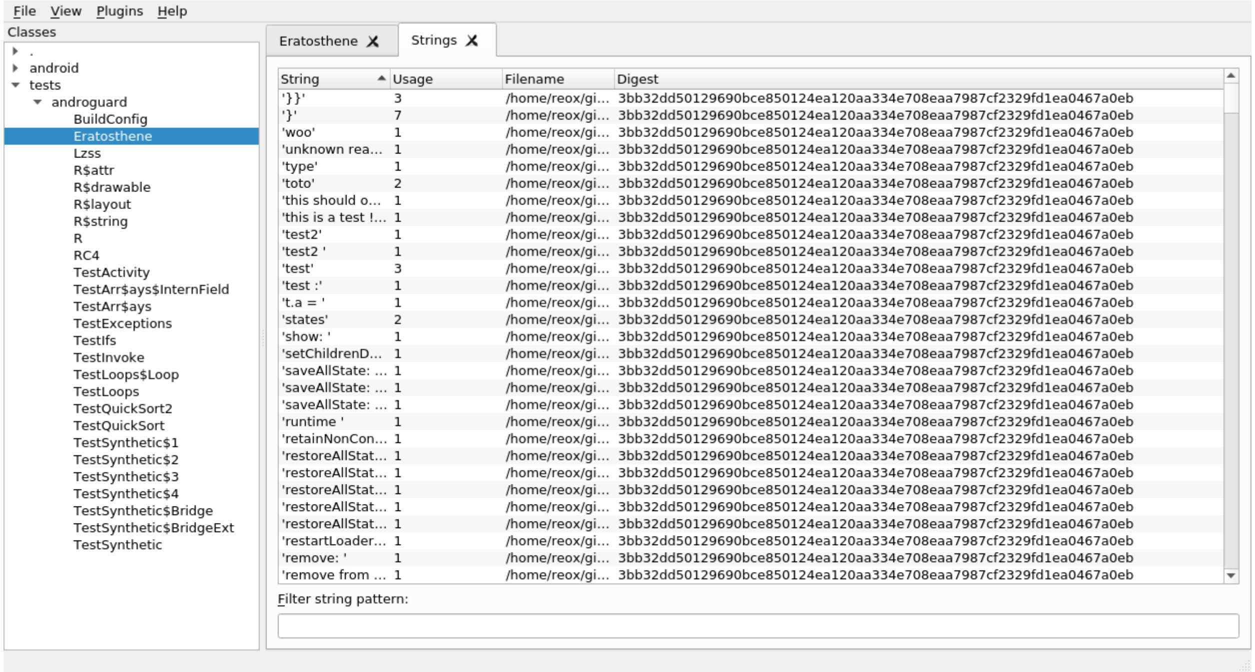
If you double click on one of the classes, you will get the disassembler view:



```
FileAddr  Disasm listing
00000000          12 08                 const/4               v8 0
00000004          12 1b                 const/4               v11 1
00000008          d8 09 0c 01           add-int/lit8          v9 v12 1
00000010          23 90 53 02           new-array             v0 v9 [Z
00000018          4e 0b 00 08           aput-boolean          v11 v0 v8
00000020          4e 0b 00 0b           aput-boolean          v11 v0 v11
00000028          83 c9                 int-to-double         v9 v12
0000002c          71 20 20 0d a9 00     invoke-static         v9 v10 Ljava/lang/Math;->sqrt(D)D
00000038          0b 09                 move-result-wide      v9
0000003c          8a 97                 double-to-int         v7 v9
00000040          12 23                 const/4               v3 2
00000044          37 73 0e 00           if-le                 v3 v7 14
0000004c          12 02                 const/4               v2 0
00000050          21 09                 array-length          v9 v0
00000054          34 98 19 00           if-lt                 v8 v9 25
0000005c          12 04                 const/4               v4 0
00000060          23 26 41 02           new-array             v6 v2 [I
00000068          12 03                 const/4               v3 0
0000006c          21 08                 array-length          v8 v0
00000070          34 83 1b 00           if-lt                 v3 v8 27
00000078          11 06                 return-object         v6
0000007c          47 09 00 03           aget-boolean          v9 v0 v3
00000084          39 09 06 00           if-nez                v9 6
0000008c          92 04 03 03           mul-int               v4 v3 v3
00000094          37 c4 05 00           if-le                 v4 v12 5
0000009c          d8 03 03 01           add-int/lit8          v3 v3 1
000000a4          28 e8                 goto                  -24
000000a8          4e 0b 00 04           aput-boolean          v11 v0 v4
000000b0          b0 34                 add-int/2addr         v4 v3
POS: 00000000   DWORD: 0D221070   QWORD: 000E00000D221070   BYTE: 70   <no selection>
```

Analysis of /home/reox/git/androguard/examples/android/TestsAndroguard/bin/TestActivity.apk done!

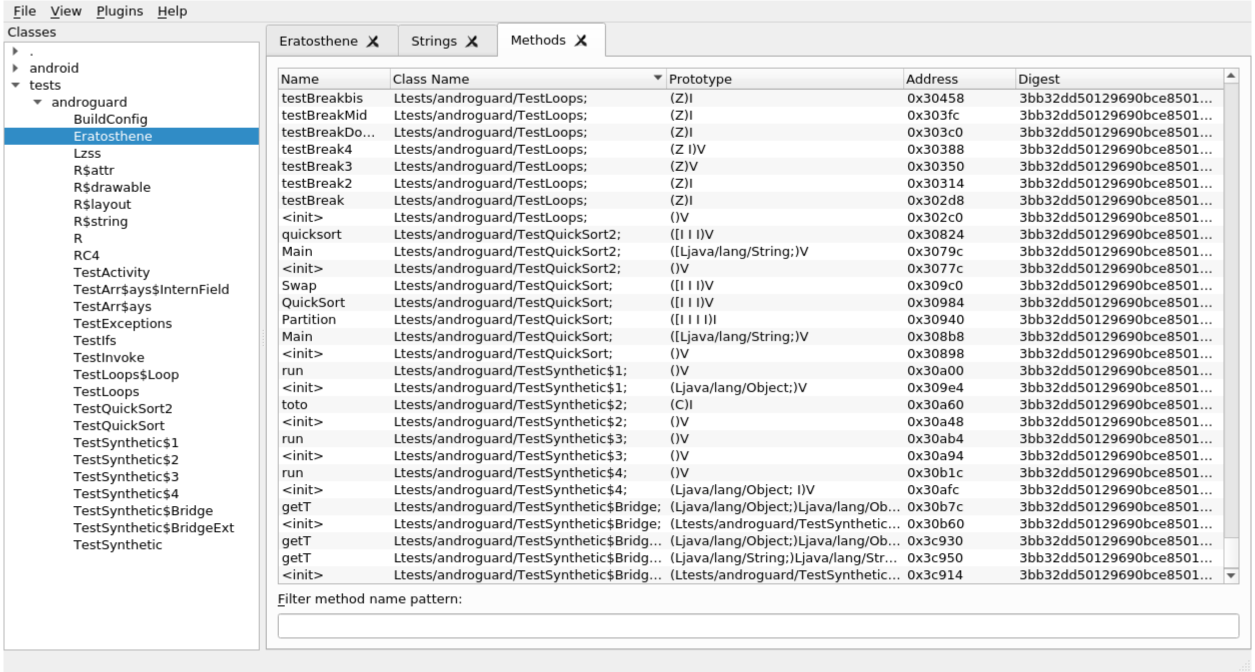
Under View, Strings you will find a list of all Strings inside the DEX file(s):



String	Usage	Filename	Digest
'}}'	3	/home/reox/gi...	3bb32dd50129690bce850124ea120aa334e708eaa7987cf2329fd1ea0467a0eb
'}'	7	/home/reox/gi...	3bb32dd50129690bce850124ea120aa334e708eaa7987cf2329fd1ea0467a0eb
'woo'	1	/home/reox/gi...	3bb32dd50129690bce850124ea120aa334e708eaa7987cf2329fd1ea0467a0eb
'unknown rea...	1	/home/reox/gi...	3bb32dd50129690bce850124ea120aa334e708eaa7987cf2329fd1ea0467a0eb
'type'	1	/home/reox/gi...	3bb32dd50129690bce850124ea120aa334e708eaa7987cf2329fd1ea0467a0eb
'toto'	2	/home/reox/gi...	3bb32dd50129690bce850124ea120aa334e708eaa7987cf2329fd1ea0467a0eb
'this should o...	1	/home/reox/gi...	3bb32dd50129690bce850124ea120aa334e708eaa7987cf2329fd1ea0467a0eb
'this is a test !...	1	/home/reox/gi...	3bb32dd50129690bce850124ea120aa334e708eaa7987cf2329fd1ea0467a0eb
'test2'	1	/home/reox/gi...	3bb32dd50129690bce850124ea120aa334e708eaa7987cf2329fd1ea0467a0eb
'test2 '	1	/home/reox/gi...	3bb32dd50129690bce850124ea120aa334e708eaa7987cf2329fd1ea0467a0eb
'test'	3	/home/reox/gi...	3bb32dd50129690bce850124ea120aa334e708eaa7987cf2329fd1ea0467a0eb
'test :'	1	/home/reox/gi...	3bb32dd50129690bce850124ea120aa334e708eaa7987cf2329fd1ea0467a0eb
't.a = '	1	/home/reox/gi...	3bb32dd50129690bce850124ea120aa334e708eaa7987cf2329fd1ea0467a0eb
'states'	2	/home/reox/gi...	3bb32dd50129690bce850124ea120aa334e708eaa7987cf2329fd1ea0467a0eb
'show: '	1	/home/reox/gi...	3bb32dd50129690bce850124ea120aa334e708eaa7987cf2329fd1ea0467a0eb
'setChildrenD...	1	/home/reox/gi...	3bb32dd50129690bce850124ea120aa334e708eaa7987cf2329fd1ea0467a0eb
'saveAllState: ...	1	/home/reox/gi...	3bb32dd50129690bce850124ea120aa334e708eaa7987cf2329fd1ea0467a0eb
'saveAllState: ...	1	/home/reox/gi...	3bb32dd50129690bce850124ea120aa334e708eaa7987cf2329fd1ea0467a0eb
'saveAllState: ...	1	/home/reox/gi...	3bb32dd50129690bce850124ea120aa334e708eaa7987cf2329fd1ea0467a0eb
'runtime '	1	/home/reox/gi...	3bb32dd50129690bce850124ea120aa334e708eaa7987cf2329fd1ea0467a0eb
'retainNonCon...	1	/home/reox/gi...	3bb32dd50129690bce850124ea120aa334e708eaa7987cf2329fd1ea0467a0eb
'restoreAllStat...	1	/home/reox/gi...	3bb32dd50129690bce850124ea120aa334e708eaa7987cf2329fd1ea0467a0eb
'restoreAllStat...	1	/home/reox/gi...	3bb32dd50129690bce850124ea120aa334e708eaa7987cf2329fd1ea0467a0eb
'restoreAllStat...	1	/home/reox/gi...	3bb32dd50129690bce850124ea120aa334e708eaa7987cf2329fd1ea0467a0eb
'restoreAllStat...	1	/home/reox/gi...	3bb32dd50129690bce850124ea120aa334e708eaa7987cf2329fd1ea0467a0eb
'restoreAllStat...	1	/home/reox/gi...	3bb32dd50129690bce850124ea120aa334e708eaa7987cf2329fd1ea0467a0eb
'restartLoader...	1	/home/reox/gi...	3bb32dd50129690bce850124ea120aa334e708eaa7987cf2329fd1ea0467a0eb
'remove: '	1	/home/reox/gi...	3bb32dd50129690bce850124ea120aa334e708eaa7987cf2329fd1ea0467a0eb
'remove from ...	1	/home/reox/gi...	3bb32dd50129690bce850124ea120aa334e708eaa7987cf2329fd1ea0467a0eb

Filter string pattern:

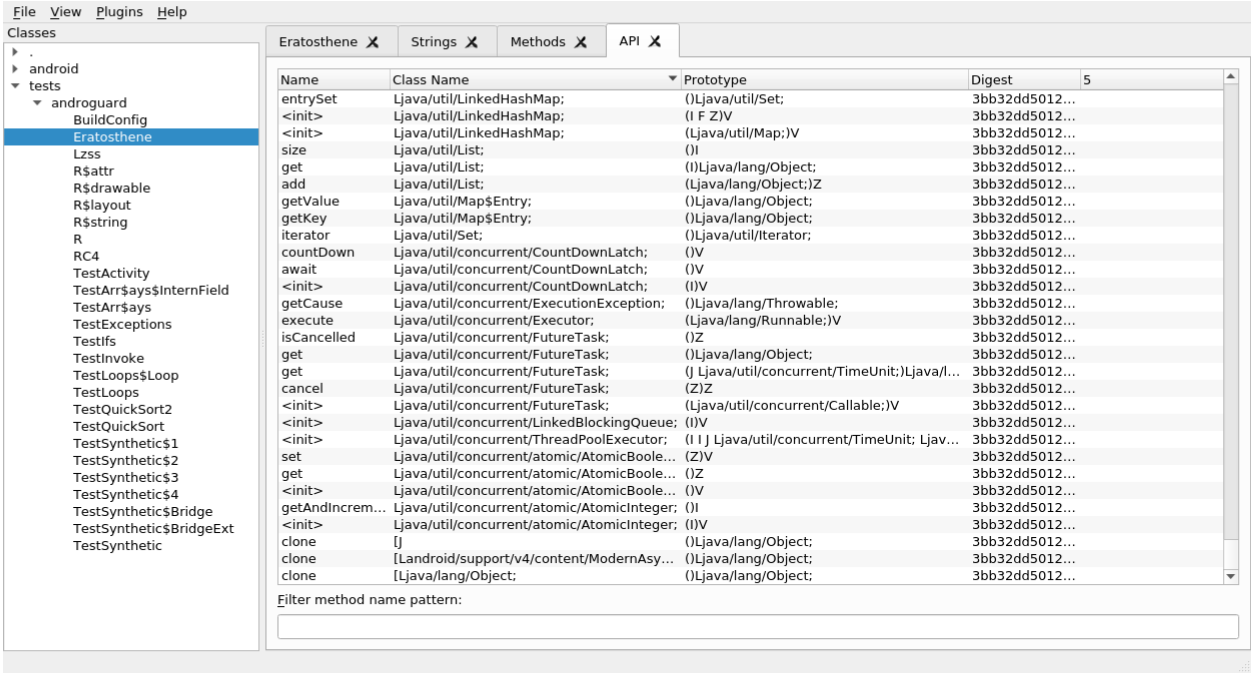
View, Methods shows all methods found in the DEX files(s):



Name	Class Name	Prototype	Address	Digest
testBreakbis	Ltests/androguard/TestLoops;	(Z)I	0x30458	3bb32dd50129690bce8501...
testBreakMid	Ltests/androguard/TestLoops;	(Z)I	0x303fc	3bb32dd50129690bce8501...
testBreakDo...	Ltests/androguard/TestLoops;	(Z)I	0x303c0	3bb32dd50129690bce8501...
testBreak4	Ltests/androguard/TestLoops;	(Z I)V	0x30388	3bb32dd50129690bce8501...
testBreak3	Ltests/androguard/TestLoops;	(Z)V	0x30350	3bb32dd50129690bce8501...
testBreak2	Ltests/androguard/TestLoops;	(Z)I	0x30314	3bb32dd50129690bce8501...
testBreak	Ltests/androguard/TestLoops;	(Z)I	0x302d8	3bb32dd50129690bce8501...
<init>	Ltests/androguard/TestLoops;	()V	0x302c0	3bb32dd50129690bce8501...
quicksort	Ltests/androguard/TestQuickSort2;	([I I I)V	0x30824	3bb32dd50129690bce8501...
Main	Ltests/androguard/TestQuickSort2;	([Ljava/lang/String;)V	0x3079c	3bb32dd50129690bce8501...
<init>	Ltests/androguard/TestQuickSort2;	()V	0x3077c	3bb32dd50129690bce8501...
Swap	Ltests/androguard/TestQuickSort;	([I I I)V	0x309c0	3bb32dd50129690bce8501...
QuickSort	Ltests/androguard/TestQuickSort;	([I I I)V	0x30984	3bb32dd50129690bce8501...
Partition	Ltests/androguard/TestQuickSort;	([I I I I)I	0x30940	3bb32dd50129690bce8501...
Main	Ltests/androguard/TestQuickSort;	([Ljava/lang/String;)V	0x308b8	3bb32dd50129690bce8501...
<init>	Ltests/androguard/TestQuickSort;	()V	0x30898	3bb32dd50129690bce8501...
run	Ltests/androguard/TestSynthetic$1;	()V	0x30a00	3bb32dd50129690bce8501...
<init>	Ltests/androguard/TestSynthetic$1;	(Ljava/lang/Object;)V	0x309e4	3bb32dd50129690bce8501...
toto	Ltests/androguard/TestSynthetic$2;	(C)I	0x30a60	3bb32dd50129690bce8501...
<init>	Ltests/androguard/TestSynthetic$2;	()V	0x30a48	3bb32dd50129690bce8501...
run	Ltests/androguard/TestSynthetic$3;	()V	0x30ab4	3bb32dd50129690bce8501...
<init>	Ltests/androguard/TestSynthetic$3;	()V	0x30a94	3bb32dd50129690bce8501...
run	Ltests/androguard/TestSynthetic$4;	()V	0x30b1c	3bb32dd50129690bce8501...
<init>	Ltests/androguard/TestSynthetic$4;	(Ljava/lang/Object; I)V	0x30afc	3bb32dd50129690bce8501...
getT	Ltests/androguard/TestSynthetic$Bridge;	(Ljava/lang/Object;)Ljava/lang/Ob...	0x30b7c	3bb32dd50129690bce8501...
<init>	Ltests/androguard/TestSynthetic$Bridge;	(Ltests/androguard/TestSynthetic...	0x30b60	3bb32dd50129690bce8501...
getT	Ltests/androguard/TestSynthetic$Bridg...	(Ljava/lang/Object;)Ljava/lang/Ob...	0x3c930	3bb32dd50129690bce8501...
getT	Ltests/androguard/TestSynthetic$Bridg...	(Ljava/lang/String;)Ljava/lang/Str...	0x3c950	3bb32dd50129690bce8501...
<init>	Ltests/androguard/TestSynthetic$Bridg...	(Ltests/androguard/TestSynthetic...	0x3c914	3bb32dd50129690bce8501...

Filter method name pattern:

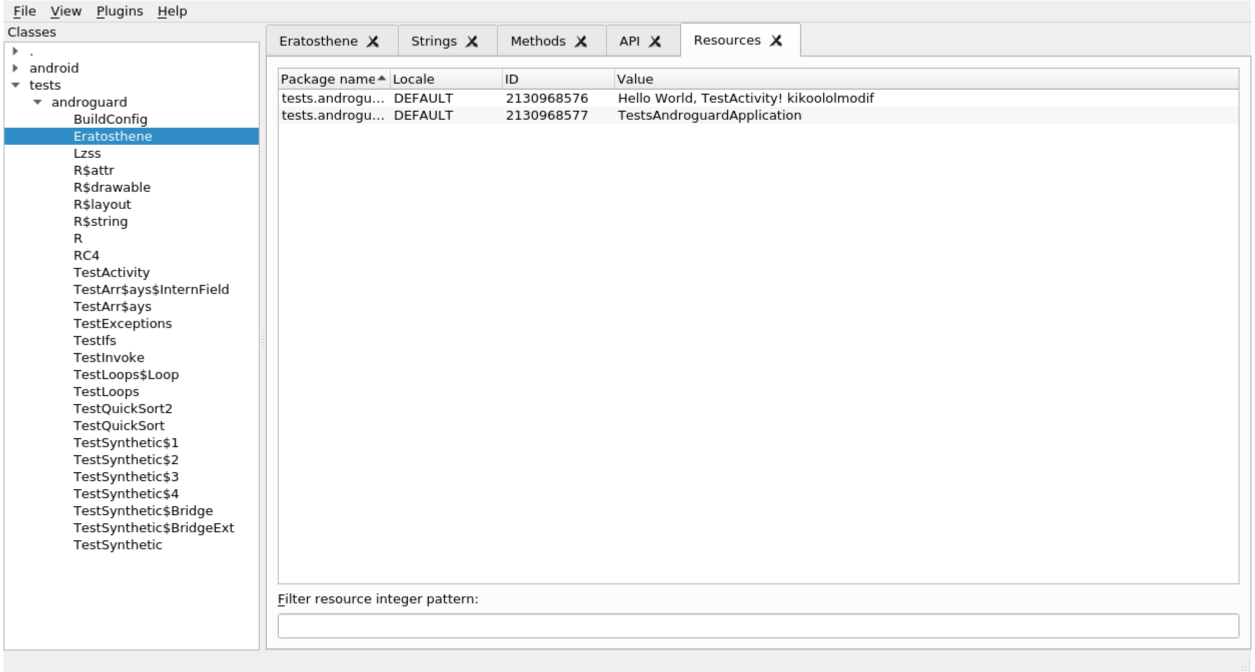
Using View, API you will get a list of all API methods (or bascically all external Methods) which are used inside the APK:



Name	Class Name	Prototype	Digest	5
entrySet	Ljava/util/LinkedHashMap;	()Ljava/util/Set;	3bb32dd5012...	
<init>	Ljava/util/LinkedHashMap;	(I F Z)V	3bb32dd5012...	
<init>	Ljava/util/LinkedHashMap;	(Ljava/util/Map;)V	3bb32dd5012...	
size	Ljava/util/List;	()I	3bb32dd5012...	
get	Ljava/util/List;	(I)Ljava/lang/Object;	3bb32dd5012...	
add	Ljava/util/List;	(Ljava/lang/Object;)Z	3bb32dd5012...	
getValue	Ljava/util/Map$Entry;	()Ljava/lang/Object;	3bb32dd5012...	
getKey	Ljava/util/Map$Entry;	()Ljava/lang/Object;	3bb32dd5012...	
iterator	Ljava/util/Set;	()Ljava/util/Iterator;	3bb32dd5012...	
countDown	Ljava/util/concurrent/CountDownLatch;	()V	3bb32dd5012...	
await	Ljava/util/concurrent/CountDownLatch;	()V	3bb32dd5012...	
<init>	Ljava/util/concurrent/CountDownLatch;	(I)V	3bb32dd5012...	
getCause	Ljava/util/concurrent/ExecutionException;	()Ljava/lang/Throwable;	3bb32dd5012...	
execute	Ljava/util/concurrent/Executor;	(Ljava/lang/Runnable;)V	3bb32dd5012...	
isCancelled	Ljava/util/concurrent/FutureTask;	()Z	3bb32dd5012...	
get	Ljava/util/concurrent/FutureTask;	()Ljava/lang/Object;	3bb32dd5012...	
get	Ljava/util/concurrent/FutureTask;	(J Ljava/util/concurrent/TimeUnit;)Ljava/l...	3bb32dd5012...	
cancel	Ljava/util/concurrent/FutureTask;	(Z)Z	3bb32dd5012...	
<init>	Ljava/util/concurrent/FutureTask;	(Ljava/util/concurrent/Callable;)V	3bb32dd5012...	
<init>	Ljava/util/concurrent/LinkedBlockingQueue;	(I)V	3bb32dd5012...	
<init>	Ljava/util/concurrent/ThreadPoolExecutor;	(I I J Ljava/util/concurrent/TimeUnit; Ljav...	3bb32dd5012...	
set	Ljava/util/concurrent/atomic/AtomicBoole...	(Z)V	3bb32dd5012...	
get	Ljava/util/concurrent/atomic/AtomicBoole...	()Z	3bb32dd5012...	
<init>	Ljava/util/concurrent/atomic/AtomicBoole...	()V	3bb32dd5012...	
getAndIncrem...	Ljava/util/concurrent/atomic/AtomicInteger;	()I	3bb32dd5012...	
<init>	Ljava/util/concurrent/atomic/AtomicInteger;	(I)V	3bb32dd5012...	
clone	[J	()Ljava/lang/Object;	3bb32dd5012...	
clone	[Landroid/support/v4/content/ModernAsy...	()Ljava/lang/Object;	3bb32dd5012...	
clone	[Ljava/lang/Object;	()Ljava/lang/Object;	3bb32dd5012...	

Filter method name pattern:

At last, you can get a list of all string resources from the *resources.arsc* file using View, Resources:



Package name	Locale	ID	Value
tests.androgu...	DEFAULT	2130968576	Hello World, TestActivity! kikoololmodif
tests.androgu...	DEFAULT	2130968577	TestsAndroguardApplication

It is possible to add other APK or DEX files at any point using File, Add. In order to save the current state of the GUI and resume later, just go to File, Save and save the file as an *.ag* file. To resume later, just open the file with File, Open again.

Plugin System

Warning: Plugins are not tested and there are no examples right now!

The androguard gui supports plugins to be loaded.

A plugin is a python file which implements the following class:

```
class PluginEntry:
    def __init__(self, session):
        """
        Session is a :class:`~androguard.session.Session` object.
        """
        self.session = session
```

1.2.5 androguard sign - Print Certificate Fingerprints

Get the fingerprints of the signing certificates inside an APK.

```
Usage: androguard sign [OPTIONS] [APK]...

  Return the fingerprint(s) of all certificates inside an APK.

Options:
  --hash [md5|sha1|sha256|sha512]
```

(continues on next page)

(continued from previous page)

```
                                  Fingerprint Hash algorithm  [default: sha1]
  -a, --all                       Print all supported hashes  [default: False]
  -s, --show                      Additionally of printing the fingerprints,
                                  show more certificate information  [default:
                                  False]
  --help                          Show this message and exit.
```

Examples

```
$ androguard sign --all files/golden-aligned-v1v2-out.apk
golden-aligned-v1v2-out.apk, package: 'android.appsecurity.cts.tinyapp'
Is signed v1: True
Is signed v2: True
Found 1 unique certificates
md5 e995a5ed7137307661f854e66901ee9e
sha1 0aa07c0f297b4ae834dc85a17eea8c2cf9380ff7
sha512␣
↪4da6e6744a4dabef192b198be13b4492b0ce97469f3ce223dd9b7e8df2ee952328e06651e5e65dd3b60ac5e3946e16cf705
sha256 fb5dbd3c669af9fc236c6991e6387b7f11ff0590997f22d0f5c74ff40e04fca8
```

1.2.6 androguard axml - AndroidManifest.xml parser

Parse the AndroidManifest.xml from an APK and show/save the XML file.

```
Usage: androguard axml [OPTIONS] [FILE_]

  Parse the AndroidManifest.xml.

  Parsing is either direct or from a given APK and prints in XML format or
  saves to file.

  This tool can also be used to process any AXML encoded file, for example
  from the layout directory.

  Example:

      $ androguard axml AndroidManifest.xml

Options:
  -i, --input FILE     AndroidManifest.xml or APK to parse (legacy option)
  -o, --output TEXT    filename to save the decoded AndroidManifest.xml to,
                       default stdout
  -r, --resource TEXT  Resource (any binary XML file) inside the APK to parse
                       instead of AndroidManifest.xml
  --help               Show this message and exit.
```

1.2.7 androguard arsc - resources.arsc parser

Parse the resources.arsc file from an APK and print human readable XML.

```
Usage: androguard arsc [OPTIONS] [FILE_]

  Decode resources.arsc either directly from a given file or from an APK.

  Example:

      $ androguard arsc app.apk

Options:
  -i, --input PATH     resources.arsc or APK to parse (legacy option)
  -o, --output TEXT    filename to save the decoded resources to
  -p, --package TEXT   Show only resources for the given package name
                       (default: the first package name found)
  -l, --locale TEXT    Show only resources for the given locale (default:
                       '\x00\x00')
  -t, --type TEXT      Show only resources of the given type (default: public)
  --id TEXT            Resolve the given ID for the given locale and package.
                       Provide the hex ID!
  -t, --list-packages  List all package names and exit
  -t, --list-locales   List all package names and exit
  -t, --list-types     List all types and exit
  --help               Show this message and exit.
```

1.2.8 androguard decompile - Decompile APKs and create CFG

androdd is a tool to create a decompiled version of an APK using the available decompilers.

Synopsis

```
Usage: androguard decompile [OPTIONS] [FILE_]

  Decompile an APK and create Control Flow Graphs.

  Example:

      $ androguard resources.arsc

Options:
  -i, --input FILE       APK to parse (legacy option)
  -o, --output TEXT      output directory. If the output folder already
                         exsist, it will be overwritten!  [required]
  -f, --format TEXT      Additionally write control flow graphs for each
                         method, specify the format for example png, jpg, raw
                         (write dot file), ...
  -j, --jar              Use DEX2JAR to create a JAR file
  -l, --limit TEXT       Limit to certain methods only by regex (default:
                         '.*')
  -d, --decompiler TEXT  Use a different decompiler (default: DAD)
  --help                 Show this message and exit.
```

It also can generate control flow graphs (CFG) for each method using the graphviz format. The CFGs can be exported as image file directly.

Additionally to the decompiled classes in .java format, each method is given in a SMALI like format (.ag files)

All filenames are sanatized, so they should work on most operating systems and filesystems.

Examples

To get all CFG in png format and limit the processing only to a certain namespace, the following command can be used:

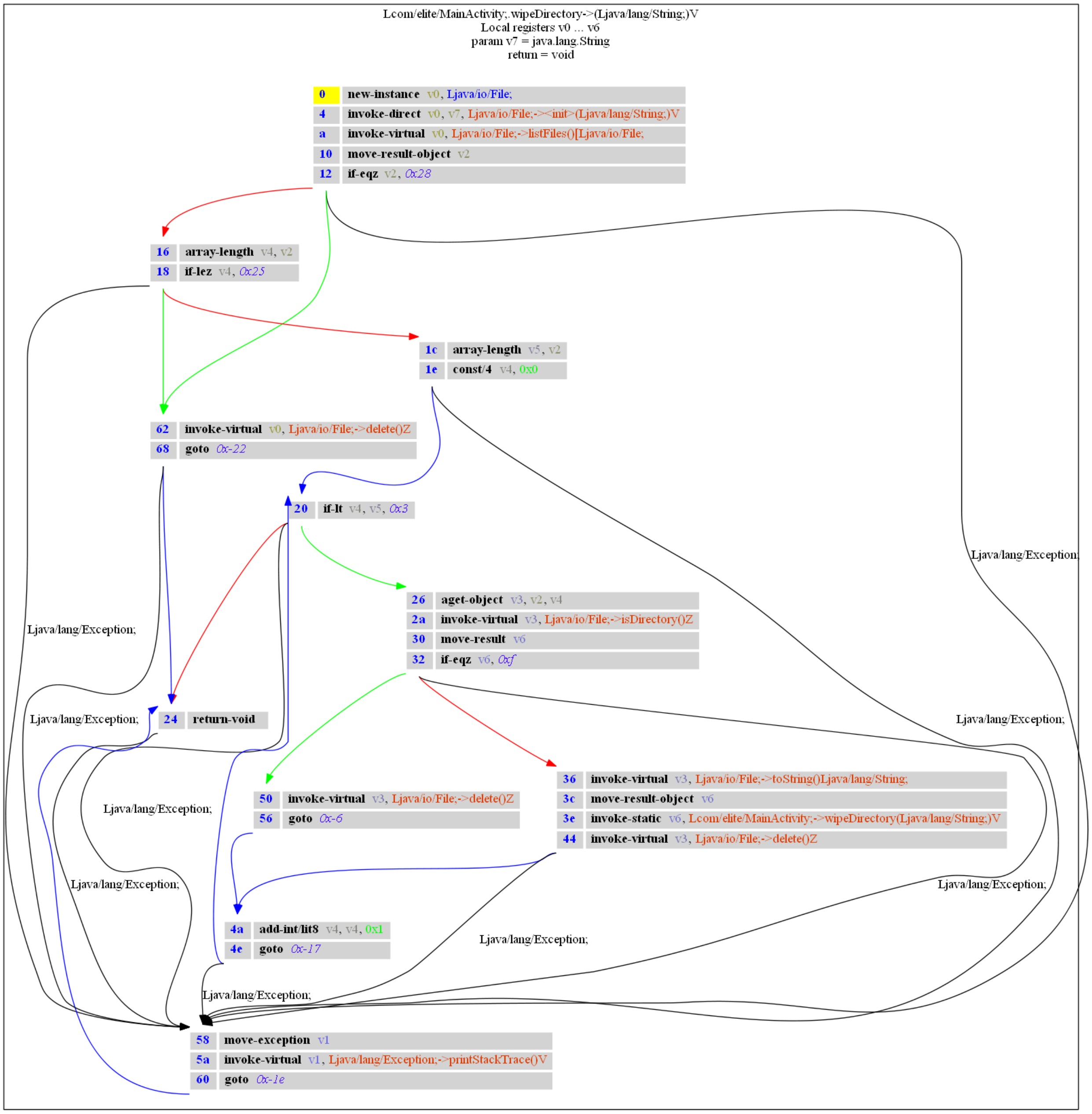
```
androguard decompile -o outputfolder -f png -i someapp.apk --limit "^Lcom/elite/.*"
```

Please make sure that graphviz and pydot are installed.

```
$ sudo apt-get install graphviz
$ pip install -U pydot
```

This will decompile the app *someapp.apk* into the folder *outputfolder* and limit the processing to all methods, where the classname starts with *com.elite.*.

A CFG might look like this:


Lcom/elite/MainActivity;-wipeDirectory->(Ljava/lang/String;)V
Local registers v0 ... v6
param v7 = java.lang.String
return = void
0 new-instance v0, Ljava/io/File;
4 invoke-direct v0, v7, Ljava/io/File;-><init>(Ljava/lang/String;)V
a invoke-virtual v0, Ljava/io/File;->listFiles()[Ljava/io/File;
10 move-result-object v2
12 if-eqz v2, 0x28
16 array-length v4, v2
18 if-lez v4, 0x25
1c array-length v5, v2
1e const/4 v4, 0x0
62 invoke-virtual v0, Ljava/io/File;->delete()Z
68 goto 0x-22
20 if-lt v4, v5, 0x3
26 aget-object v3, v2, v4
2a invoke-virtual v3, Ljava/io/File;->isDirectory()Z
30 move-result v6
32 if-eqz v6, 0xf
24 return-void
50 invoke-virtual v3, Ljava/io/File;->delete()Z
56 goto 0x-6
36 invoke-virtual v3, Ljava/io/File;->toString()Ljava/lang/String;
3c move-result-object v6
3e invoke-static v6, Lcom/elite/MainActivity;->wipeDirectory(Ljava/lang/String;)V
44 invoke-virtual v3, Ljava/io/File;->delete()Z
4a add-int/lit8 v4, v4, 0x1
4e goto 0x-17
58 move-exception v1
5a invoke-virtual v1, Ljava/lang/Exception;->printStackTrace()V
60 goto 0x-1e
Ljava/lang/Exception;
Ljava/lang/Exception;
Ljava/lang/Exception;
Ljava/lang/Exception;
Ljava/lang/Exception;
Ljava/lang/Exception;
Ljava/lang/Exception;
Ljava/lang/Exception;
Ljava/lang/Exception;

while the *.ag* file has this content:

```
# Lcom/elite/MainActivity;->wipeDirectory(Ljava/lang/String;)V [access_flags=private static]
#
# Parameters:
# - local registers: v0...v6
# - v7:java.lang.String
#
# - return:void

wipeDirectory-BB@0x0 : [ wipeDirectory-BB@0x16 wipeDirectory-BB@0x62 ]
    0        (00000000) new-instance          v0, Ljava/io/File;
```

(continues on next page)

(continued from previous page)

```
    1       (00000004) invoke-direct        v0, v7, Ljava/io/File;-><init>(Ljava/lang/
↪String;)V
    2       (0000000a) invoke-virtual       v0, Ljava/io/File;->listFiles()[Ljava/io/
↪File;
    3       (00000010) move-result-object   v2
    4       (00000012) if-eqz               v2, +28
    0:55
    (Ljava/lang/Exception; -> 58 wipeDirectory-BB@0x58)

wipeDirectory-BB@0x16 : [ wipeDirectory-BB@0x1c wipeDirectory-BB@0x62 ]
    5       (00000016) array-length         v4, v2
    6       (00000018) if-lez               v4, +25
    0:55
    (Ljava/lang/Exception; -> 58 wipeDirectory-BB@0x58)

wipeDirectory-BB@0x1c : [ wipeDirectory-BB@0x20 ]
    7       (0000001c) array-length         v5, v2
    8       (0000001e) const/4              v4, 0
    0:55
    (Ljava/lang/Exception; -> 58 wipeDirectory-BB@0x58)

wipeDirectory-BB@0x20 : [ wipeDirectory-BB@0x24 wipeDirectory-BB@0x26 ]
    9       (00000020) if-lt                v4, v5, +3
    0:55
    (Ljava/lang/Exception; -> 58 wipeDirectory-BB@0x58)

wipeDirectory-BB@0x24 :
    10      (00000024) return-void
    0:55
    (Ljava/lang/Exception; -> 58 wipeDirectory-BB@0x58)

wipeDirectory-BB@0x26 : [ wipeDirectory-BB@0x36 wipeDirectory-BB@0x50 ]
    11      (00000026) aget-object          v3, v2, v4
    12      (0000002a) invoke-virtual       v3, Ljava/io/File;->isDirectory()Z
    13      (00000030) move-result          v6
    14      (00000032) if-eqz               v6, +f
    0:55
    (Ljava/lang/Exception; -> 58 wipeDirectory-BB@0x58)

wipeDirectory-BB@0x36 : [ wipeDirectory-BB@0x4a ]
    15      (00000036) invoke-virtual       v3, Ljava/io/File;->toString()Ljava/lang/
↪String;
    16      (0000003c) move-result-object   v6
    17      (0000003e) invoke-static        v6, Lcom/elite/MainActivity;->
↪wipeDirectory(Ljava/lang/String;)V
    18      (00000044) invoke-virtual       v3, Ljava/io/File;->delete()Z
    0:55
    (Ljava/lang/Exception; -> 58 wipeDirectory-BB@0x58)

wipeDirectory-BB@0x4a : [ wipeDirectory-BB@0x20 ]
    19      (0000004a) add-int/lit8         v4, v4, 1
    20      (0000004e) goto                 -17
    0:55
    (Ljava/lang/Exception; -> 58 wipeDirectory-BB@0x58)

wipeDirectory-BB@0x50 : [ wipeDirectory-BB@0x4a ]
    21      (00000050) invoke-virtual       v3, Ljava/io/File;->delete()Z
```

(continues on next page)

(continued from previous page)

```
    22       (00000056) goto                 -6

wipeDirectory-BB@0x58 : [ wipeDirectory-BB@0x24 ]
    23       (00000058) move-exception       v1
    24       (0000005a) invoke-virtual       v1, Ljava/lang/Exception;->
↪printStackTrace()V
    25       (00000060) goto                 -1e

wipeDirectory-BB@0x62 : [ wipeDirectory-BB@0x24 ]
    26       (00000062) invoke-virtual       v0, Ljava/io/File;->delete()Z
    27       (00000068) goto                 -22
    62:67
    (Ljava/lang/Exception; -> 58 wipeDirectory-BB@0x58)
```

1.2.9 androguard dissassemble - Disassembler for DEX

androdis is a disassembler for DEX files.

```
Usage: androguard disassemble [OPTIONS] DEX

  Disassemble Dalvik Code with size SIZE starting from an offset

Options:
  -o, --offset INTEGER  Offset to start dissassembly inside the file
  -s, --size INTEGER    Number of bytes from offset to disassemble, 0 for
                        whole file
  --help                Show this message and exit.
```

CHAPTER

TWO

COMMONLY USED APIS

This is a just a selection of the most important top level API classes.

APK parser `androguard.core.bytecodes.apk.APK`

DEX parser `androguard.core.bytecodes.dvm.DalvikVMFormat`

AXML parser `androguard.core.bytecodes.axml.AXMLPrinter`

ARSC parser `androguard.core.bytecodes.axml.ARSCParser`

Analysis `androguard.core.analysis.analysis.Analysis`

Session `androguard.session.Session`

Automated Analysis `androguard.core.analysis.auto.AndroAuto`

Decompilers `androguard.decompiler.decompiler`

CHAPTER

THREE

COMPLETE PYTHON API

3.1 androguard package

3.1.1 Subpackages

androguard.core package

Subpackages

androguard.core.analysis package

The analysis module implements an abstraction layer for *androguard.core.bytecodes.dvm.DalvikVMFormat* objects. The the help of the `androguard.core.analysis.analysis.Analsyis` object, you can bundle several DEX files together. This is not only useful for multidex files, but also for a single dex, as Analysis offers many features to investigate DEX files. One of these features is crossreferencing (XREF). It allows you to build a graph of the methods inside the DEX files. You can then create callgraphs or find methods which use a specific API method.

Submodules

androguard.core.analysis.analysis module

class `androguard.core.analysis.analysis.`**`Analysis`**(*vm=None*)

Bases: `object`

`add`(*vm*)

Add a DalvikVMFormat to this Analysis.

Parameters vm (`androguard.core.bytecodes.dvm.DalvikVMFormat`) – `dvm.DalvikVMFormat` to add to this Analysis

`create_ipython_exports`()

> **Warning:** this feature is experimental and is currently not enabled by default! Use with caution!

Creates attributes for all classes, methods and fields on the Analysis object itself. This makes it easier to work with Analysis module in an iPython shell.

Classes can be search by typing `dx.CLASS_<tab>`, as each class is added via this attribute name. Each class will have all methods attached to it via `dx.CLASS_Foobar.METHOD_<tab>`. Fields have a similar syntax: `dx.CLASS_Foobar.FIELD_<tab>`.

As Strings can contain nearly anything, use `find_strings()` instead.

- Each *CLASS_* item will return a `ClassAnalysis`
- Each *METHOD_* item will return a `MethodAnalysis`
- Each *FIELD_* item will return a `FieldAnalysis`

create_xref()

Create Class, Method, String and Field crossreferences for all classes in the Analysis.

If you are using multiple DEX files, this function must be called when all DEX files are added. If you call the function after every DEX file, it will only work for the first time.

find_classes(*name='.*'*, *no_external=False*)

Find classes by name, using regular expression This method will return all ClassAnalysis Object that match the name of the class.

Parameters

- **name** – regular expression for class name (default ".*")
- **no_external** – Remove external classes from the output (default False)

Return type Iterator[*ClassAnalysis*]

find_fields(*classname='.*'*, *fieldname='.*'*, *fieldtype='.*'*, *accessflags='.*'*)

find fields by regex

Parameters

- **classname** – regular expression of the classname
- **fieldname** – regular expression of the fieldname
- **fieldtype** – regular expression of the fieldtype
- **accessflags** – regular expression of the access flags

Return type Iterator[*FieldAnalysis*]

find_methods(*classname='.*'*, *methodname='.*'*, *descriptor='.*'*, *accessflags='.*'*, *no_external=False*)

Find a method by name using regular expression. This method will return all MethodAnalysis objects, which match the classname, methodname, descriptor and accessflags of the method.

Parameters

- **classname** – regular expression for the classname
- **methodname** – regular expression for the method name
- **descriptor** – regular expression for the descriptor
- **accessflags** – regular expression for the accessflags
- **no_external** – Remove external method from the output (default False)

Return type Iterator[*MethodAnalysis*]

find_strings(*string='.*'*)

Find strings by regex

Parameters **string** – regular expression for the string to search for

Return type Iterator[*StringAnalysis*]

get_call_graph(*classname='.*', methodname='.*', descriptor='.*', accessflags='.*', no_isolated=False, entry_points=[]*)

Generate a directed graph based on the methods found by the filters applied. The filters are the same as in `find_methods()`

A networkx.MultiDiGraph is returned, containing all xrefs. That means a method which calls another method multiple times, will have multiple edges between them. Attached to the edge is the attribute *offset*, which gives the code offset inside the method of the call.

Specifying filters will not remove the methods if they are called by some other method.

The callgraph will check for both directions of edges. Thus, if you specify a single class as input, it will contain all classes which are called by this class (xref_to), as well as all methods who calls the specified one (xref_from).

Each node will contain the following meta information as attribute:

- external: is the method external or not (boolean)
- entrypoint: is the method a known entry point (boolean)
- native: is the method a native method by signature (boolean)
- public: is the method declared public (boolean)
- static: is the method declared static (boolean)
- vm: An ID of the DEX file where this method is declared or 0 if external (signed int)
- codesize: size of code of the method or zero if external (int)

Parameters

- **classname** – regular expression of the classname (default: ".*")
- **methodname** – regular expression of the methodname (default: ".*")
- **descriptor** – regular expression of the descriptor (default: ".*")
- **accessflags** – regular expression of the access flags (default: ".*")
- **no_isolated** – remove isolated nodes from the graph, e.g. methods which do not call anything (default: False)
- **entry_points** – A list of classes that are marked as entry point

Return type networkx.MultiDiGraph

get_class_analysis(*class_name*)

Returns the `ClassAnalysis` object for a given classname.

Parameters **class_name** – classname like 'Ljava/lang/Object;' (including L and ;)

Returns `ClassAnalysis`

Return type *ClassAnalysis*

get_classes()

Returns a list of `ClassAnalysis` objects

Returns both internal and external classes (if any)

Return type Iterator[*ClassAnalysis*]

get_external_classes()

Returns all external classes, that means all classes that are not defined in the given set of *DalvikVMObjects*.

Return type Iterator[*ClassAnalysis*]

get_field_analysis(*field*)

Get the FieldAnalysis for a given fieldname

Parameters **field** (`androguard.core.bytecodes.dvm.EncodedField`) – the field

Returns *`FieldAnalysis`*

Return type *FieldAnalysis*

get_fields()

Returns a list of *FieldAnalysis* objects

Return type Iterator[*FieldAnalysis*]

get_internal_classes()

Returns all external classes, that means all classes that are defined in the given set of `DalvikVMFormat`.

Return type Iterator[*ClassAnalysis*]

get_method(*method*)

Get the *`MethodAnalysis`* object for a given `EncodedMethod`. This Analysis object is used to enhance EncodedMethods.

Parameters **method** – `EncodedMethod` to search for

Returns *`MethodAnalysis`* object for the given method, or None if method was not found

Return type *MethodAnalysis*

get_method_analysis(*method*)

Get the *`MethodAnalysis`* object for a given `EncodedMethod`. This Analysis object is used to enhance EncodedMethods.

Parameters **method** – `EncodedMethod` to search for

Returns *`MethodAnalysis`* object for the given method, or None if method was not found

Return type *MethodAnalysis*

get_method_analysis_by_name(*class_name*, *method_name*, *method_descriptor*)

Returns the crossreferencing object for a given method.

This function is similar to *`get_method_analysis()`*, with the difference that you can look up the Method by name

Parameters

- **class_name** – name of the class, for example *'Ljava/lang/Object;'*
- **method_name** – name of the method, for example *'onCreate'*
- **method_descriptor** – method descriptor, for example *'(I I)V'*

Returns *`MethodAnalysis`*

Return type *MethodAnalysis*

get_method_by_name(*class_name*, *method_name*, *method_descriptor*)

Search for a `EncodedMethod` in all classes in this analysis

Parameters

- **class_name** – name of the class, for example 'Ljava/lang/Object;'
- **method_name** – name of the method, for example 'onCreate'
- **method_descriptor** – descriptor, for example '(I I Ljava/lang/String)V

Returns EncodedMethod or None if method was not found

Return type *androguard.core.bytecodes.dvm.EncodedMethod*

get_methods()

Returns a list of *MethodAnalysis* objects

Return type Iterator[*MethodAnalysis*]

get_permission_usage(*permission*, *apilevel=None*)

Find the usage of a permission inside the Analysis.

example:: from androguard.misc import AnalyzeAPK a, d, dx = AnalyzeAPK("somefile.apk")

for meth in dx.get_permission_usage('android.permission.SEND_SMS', a.get_effective_target_sdk_version()): print("Using API method {}".format(meth)) print("used in:") for _, m, _ in meth.get_xref_from():

print(m.full_name)

Note: The permission mappings might be incomplete! See also *get_permissions()*.

Parameters

- **permission** – the name of the android permission (usually 'android.permission.XXX')
- **apilevel** – the requested API level or None for default

Returns yields *MethodAnalysis* objects for all using API methods

get_permissions(*apilevel=None*)

Returns the permissions and the API method based on the API level specified. This can be used to find usage of API methods which require a permission. Should be used in combination with an *APK*.

The returned permissions are a list, as some API methods require multiple permissions at once.

The following example shows the usage and how to get the calling methods using XREF:

example:: from androguard.misc import AnalyzeAPK a, d, dx = AnalyzeAPK("somefile.apk")

for meth, perm in dx.get_permissions(a.get_effective_target_sdk_version()): print("Using API method {} for permission {}".format(meth, perm)) print("used in:") for _, m, _ in meth.get_xref_from():

print(m.full_name)

..note:: This method might be unreliable and might not extract all used permissions. The permission mapping is based on [Axplorer](https://github.com/reddr/axplorer) and might be incomplete due to the nature of the extraction process. Unfortunately, there is no official API<->Permission mapping.

The output of this method relies also on the set API level. If the wrong API level is used, the results might be wrong.

Parameters **apilevel** – API level to load, or None for default

Returns yields tuples of *MethodAnalysis* (of the API method) and list of permission string

get_strings()

Returns a list of `StringAnalysis` objects

Return type Iterator[*StringAnalysis*]

get_strings_analysis()

Returns a dictionary of strings and their corresponding `StringAnalysis`

Return type Dict[str, *StringAnalysis*]

is_class_present(*class_name*)

Checks if a given class name is part of this Analysis.

Parameters **class_name** – classname like 'Ljava/lang/Object;' (including L and ;)

Returns True if class was found, False otherwise

Return type bool

class androguard.core.analysis.analysis.**BasicBlocks**

Bases: `object`

This class represents all basic blocks of a method.

It is a collection of many `DVMBasicBlock`.

get()

Returns yields each basic block (`DVMBasicBlock` object)

Return type Iterator[*DVMBasicBlock*]

get_basic_block(*idx*)

get_basic_block_pos(*item*)

Get the basic block at the index

Parameters **item** – index

Returns The basic block

Return type *DVMBasicBlock*

gets()

Returns a list of basic blocks (`DVMBasicBlock` objects)

pop(*idx*)

push(*bb*)

Adds another basic block to the collection

Parameters **bb** (*DVBMBasicBlock*) – the DVMBasicBlock to add

class androguard.core.analysis.analysis.**ClassAnalysis**(*classobj*)

Bases: `object`

add_field_xref_read(*method*, *classobj*, *field*, *off*)

Add a Field Read to this class

Parameters

- **method** (`MethodAnalysis`) –
- **classobj** (`ClassAnalysis`) –
- **field** (*androguard.code.bytecodes.dvm.EncodedField*) –
- **off** (*int*) –

Returns

add_field_xref_write (*method*, *classobj*, *field*, *off*)

Add a Field Write to this class in a given method

Parameters

- **method** (MethodAnalysis) –
- **classobj** (ClassAnalysis) –
- **field** (androguard.core.bytecodes.dvm.EncodedField) –
- **off** (*int*) –

Returns

add_method (*method_analysis*)

Add the given method to this analyis. usually only called during Analysis.add and Analysis._resolve_method

Parameters **method_analysis** (MethodAnalysis) –

add_method_xref_from (*method1*, *classobj*, *method2*, *offset*)

Parameters

- **method1** (MethodAnalysis) –
- **classobj** (ClassAnalysis) –
- **method2** (MethodAnalysis) –
- **offset** (*int*) –

add_method_xref_to (*method1*, *classobj*, *method2*, *offset*)

Parameters

- **method1** (MethodAnalysis) – the calling method
- **classobj** (ClassAnalysis) – the calling class
- **method2** (MethodAnalysis) – the called method
- **offset** (*int*) – offset in the bytecode of calling method

add_xref_from (*ref_kind*, *classobj*, *methodobj*, *offset*)

Creates a crossreference from this class. XrefFrom means, that the current class is called by another class.

Parameters

- **ref_kind** (REF_TYPE) – type of call
- **classobj** (ClassAnalysis) – *ClassAnalysis* object to link
- **methodobj** (MethodAnalysis) –
- **offset** (*int*) – Offset in the methods bytecode, where the call happens

Returns

add_xref_to (*ref_kind*, *classobj*, *methodobj*, *offset*)

Creates a crossreference to another class. XrefTo means, that the current class calls another class. The current class should also be contained in the another class' XrefFrom list.

> **Warning:** The implementation of this specific method might not be what you expect! the parameter `methodobj` is the source method and not the destination in the case that `ref_kind` is const-class or new-instance!

Parameters

- **`ref_kind`** (`REF_TYPE`) – type of call
- **`classobj`** (`ClassAnalysis`) – *`ClassAnalysis`* object to link
- **`methodobj`** (`MethodAnalysis`) –
- **`offset`** (*`int`*) – Offset in the Methods Bytecode, where the call happens

Returns

`property extends`

Return the parent class

For external classes, this is not sure, thus we return always Object (which is the parent of all classes)

Returns a string of the parent class name

`get_class`()

Returns the original Dalvik VM class or the external class object.

Returns

Return type Union[*androguard.core.bytecodes.dvm.ClassDefItem*, *ExternalClass*]

`get_field_analysis`(*field*)

`get_fields`()

Return all *FieldAnalysis* objects of this class

`get_method_analysis`(*method*)

Return the MethodAnalysis object for a given EncodedMethod

Parameters **`method`** – `EncodedMethod`

Returns *`MethodAnalysis`*

Return type *MethodAnalysis*

`get_methods`()

Return all *`MethodAnalysis`* objects of this class

Return type Iterator[*MethodAnalysis*]

`get_nb_methods`()

Get the number of methods in this class

`get_vm_class`()

Returns the original Dalvik VM class or the external class object.

Returns

Return type Union[*androguard.core.bytecodes.dvm.ClassDefItem*, *ExternalClass*]

`get_xref_from`()

Returns a dictionary of all classes calling the current class. This dictionary contains also information from which method the class is accessed.

Note: this method might contains wrong information about class usage!

The dictionary contains the classes as keys (stored as *ClassAnalysis*) and has a tuple as values, where the first item is the ref_kind (which is an Enum of type *REF_TYPE*), the second one is the method in which the class is called (*MethodAnalysis*) and the third the offset in the method where the call is originating.

example:: # dx is an Analysis object for cls in dx.find_classes('.*some/name.*'):

> print("Found class {} in Analysis".format(cls.name) for caller, refs in cls.get_xref_from().items():
>
> > print(" called from {}".format(caller.name)) for ref_kind, ref_method, ref_offset in refs:
> >
> > > print(" in method {} {}".format(ref_kind, ref_method))

Return type Iterator[Tuple[*REF_TYPE*, *MethodAnalysis*, int]]

get_xref_to()

Returns a dictionary of all classes which are called by the current class. This dictionary contains also information about the method which is called.

Note: this method might contains wrong information about class usage!

The dictionary contains the classes as keys (stored as *ClassAnalysis*) and has a tuple as values, where the first item is the ref_kind (which is an Enum of type *REF_TYPE*), the second one is the method called (*MethodAnalysis*) and the third the offset in the method where the call is originating.

example:: # dx is an Analysis object for cls in dx.find_classes('.*some/name.*'):

> print("Found class {} in Analysis".format(cls.name) for calling, refs in cls.get_xref_from().items():
>
> > print(" calling class {}".format(calling.name)) for ref_kind, ref_method, ref_offset in refs:
> >
> > > print(" calling method {} {}".format(ref_kind, ref_method))

Return type Iterator[Tuple[*REF_TYPE*, *MethodAnalysis*, int]]

property implements

Get a list of interfaces which are implemented by this class

Returns a list of Interface names

is_android_api()

Tries to guess if the current class is an Android API class.

This might be not very precise unless an apilist is given, with classes that are in fact known APIs. Such a list might be generated by using the android.jar files.

Returns boolean

is_external()

Tests if this class is an external class

Returns True if the Class is external, False otherwise

property name
Return the class name

Returns

class androguard.core.analysis.analysis.**DVMBasicBlock**(*start*, *vm*, *method*, *context*)
Bases: object

A simple basic block of a dalvik method.

A basic block consists of a series of *Instruction* which are not interrupted by branch or jump instructions such as *goto*, *if*, *throw*, *return*, *switch* etc.

add_note(*note*)

clear_notes()

get_end()
Get the end offset of this basic block

Returns end offset

Return type int

get_exception_analysis()

get_instructions()
Get all instructions from a basic block.

Returns Return all instructions in the current basic block

get_last()
Get the last instruction in the basic block

Returns androguard.core.bytecodes.dvm.Instruction

get_last_length()

get_method()
Returns the originiating method

Returns the method

Return type *androguard.core.bytecodes.dvm.EncodedMethod*

get_name()

get_nb_instructions()

get_next()
Get next basic blocks

Returns a list of the next basic blocks

Return type *DVMBasicBlock*

get_notes()

get_prev()
Get previous basic blocks

Returns a list of the previous basic blocks

Return type *DVMBasicBlock*

get_special_ins(*idx*)
Return the associated instruction to a specific instruction (for example a packed/sparse switch)

Parameters **idx** – the index of the instruction

Return type None or an Instruction

get_start()

Get the starting offset of this basic block

Returns starting offset

Return type int

push(*i*)

set_childs(*values*)

set_exception_analysis(*exception_analysis*)

set_fathers(*f*)

set_notes(*value*)

show()

class androguard.core.analysis.analysis.**ExceptionAnalysis**(*exception*, *bb*)

Bases: object

get()

show_buff()

class androguard.core.analysis.analysis.**Exceptions**

Bases: object

add(*exceptions*, *basic_blocks*)

get()

get_exception(*addr_start*, *addr_end*)

gets()

class androguard.core.analysis.analysis.**ExternalClass**(*name*)

Bases: object

The ExternalClass is used for all classes that are not defined in the DEX file, thus are external classes.

Parameters **name** – Name of the external class

add_method(*method*)

get_methods()

Return the stored methods for this external class :return:

get_name()

Returns the name of the ExternalClass object

class androguard.core.analysis.analysis.**ExternalMethod**(*class_name*, *name*, *descriptor*)

Bases: object

ExternalMethod is a stub class for methods that are not part of the current Analysis. There are two possibilities for this:

1) The method is defined inside another DEX file which was not loaded into the Analysis

2) The method is an API method, hence it is defined in the Android system

External methods should have a similar API to `EncodedMethod` but obviously they have no code attached. The only known information about such methods are the class name, the method name and its descriptor.

Parameters

- **class_name** (*str*) – name of the class
- **name** (*str*) – name of the method
- **descriptor** (*List[str]*) – descriptor string

property full_name
Returns classname + name + descriptor, separated by spaces (no access flags)

get_access_flags_string()
Returns the access flags string.

Right now, this is always an empty strings, as we can not say what kind of access flags an external method might have.

get_class_name()

get_descriptor()

get_name()

property permission_api_name
Returns a name which can be used to look up in the permission maps

class androguard.core.analysis.analysis.**FieldAnalysis**(*field*)
Bases: object

add_xref_read(*classobj*, *methodobj*, *offset*)

Parameters

- **classobj** (ClassAnalysis) –
- **methodobj** (MethodAnalysis) –
- **offset** (*int*) – offset in the bytecode

add_xref_write(*classobj*, *methodobj*, *offset*)

Parameters

- **classobj** (ClassAnalysis) –
- **methodobj** (MethodAnalysis) –
- **offset** (*int*) – offset in the bytecode

get_field()
Returns the actual field object

Return type *androguard.core.bytecodes.dvm.EncodedField*

get_xref_read(*withoffset=False*)
Returns a list of xrefs where the field is read.

The list contains tuples of the originating class and methods, where the class is represented as a *ClassAnalysis*, while the method is a *MethodAnalysis*.

Parameters withoffset (*bool*) – return the xrefs including the offset

get_xref_write(*withoffset=False*)
Returns a list of xrefs where the field is written to.

The list contains tuples of the originating class and methods, where the class is represented as a *ClassAnalysis*, while the method is a *MethodAnalysis*.

Parameters **withoffset** (*bool*) – return the xrefs including the offset

property name

class androguard.core.analysis.analysis.**MethodAnalysis**(*vm*, *method*)

Bases: object

This class analyses in details a method of a class/dex file It is a wrapper around a EncodedMethod and enhances it by using multiple *DVMBasicBlock* encapsulated in a *BasicBlocks* object.

property access

Returns the access flags to the method as a string

add_xref_from(*classobj*, *methodobj*, *offset*)

Add a crossrefernece from another method (this method is called by another method)

Parameters

- **classobj** – *ClassAnalysis*
- **methodobj** – *EncodedMethod*
- **offset** – integer where in the method the call happens

add_xref_to(*classobj*, *methodobj*, *offset*)

Add a crossreference to another method (this method calls another method)

Parameters

- **classobj** – *ClassAnalysis*
- **methodobj** – *EncodedMethod*
- **offset** – integer where in the method the call happens

property class_name

Returns the name of the class of this method

property descriptor

Returns the type descriptor for this method

property full_name

Returns classname + name + descriptor, separated by spaces (no access flags)

get_access_flags_string()

Returns the concatenated access flags string

get_basic_blocks()

Returns the *BasicBlocks* generated for this method. The *BasicBlocks* can be used to get a control flow graph (CFG) of the method.

Return type a *BasicBlocks* object

get_class_name()

Return the class name of the method

get_length()

Returns an integer which is the length of the code

Return type int

get_method()

Return type *androguard.core.bytecodes.dvm.EncodedMethod*

Returns

get_vm()

Return type *androguard.core.bytecodes.dvm.DalvikVMFormat*

Returns

get_xref_from()

Returns a list of tuples containing the class, method and offset of the call, from where this object was called.

The list of tuples has the form: (*ClassAnalysis*, *EncodedMethod* or *ExternalMethod*, int)

get_xref_to()

Returns a list of tuples containing the class, method and offset of the call, which are called by this method.

The list of tuples has the form: (*ClassAnalysis*, *EncodedMethod* or *ExternalMethod*, int)

is_android_api()

Returns True if the method seems to be an Android API method.

This method might be not very precise unless an list of known API methods is given.

Returns boolean

is_external()

Returns True if the underlying method is external

Return type boolean

property name

Returns the name of this method

show()

Prints the content of this method to stdout.

This will print the method signature and the decompiled code.

show_xrefs()

class androguard.core.analysis.analysis.**MethodClassAnalysis**(*meth*)

Bases: *androguard.core.analysis.analysis.MethodAnalysis*

Deprecated since version 3.4.0: Always use MethodAnalysis! This method is just here for compatability

class androguard.core.analysis.analysis.**REF_TYPE**

Bases: enum.IntEnum

Stores the opcodes for the type of usage in an XREF.

Used in *ClassAnalysis* to store the type of reference to the class.

INVOKE_DIRECT = 112

INVOKE_DIRECT_RANGE = 118

INVOKE_INTERFACE = 114

INVOKE_INTERFACE_RANGE = 120

INVOKE_STATIC = 113

INVOKE_STATIC_RANGE = 119

INVOKE_SUPER = 111

INVOKE_SUPER_RANGE = 117

INVOKE_VIRTUAL = 110

INVOKE_VIRTUAL_RANGE = 116

REF_CLASS_USAGE = 28

REF_NEW_INSTANCE = 34

class androguard.core.analysis.analysis.**StringAnalysis**(*value*)

Bases: object

StringAnalysis contains the XREFs of a string.

As Strings are only used as a source, they only contain the XREF_FROM set, i.e. where the string is used.

This Array stores the information in which method the String is used.

add_xref_from(*classobj*, *methodobj*, *off*)

Adds a xref from the given method to this string

Parameters

- **classobj** (ClassAnalysis) –
- **methodobj** (MethodAnalysis) –
- **off** (*int*) – offset in the bytecode of the call

get_orig_value()

Return the original, read only, value of the String

Returns the original value

get_value()

Return the (possible overwritten) value of the String

Returns the value of the string

get_xref_from(*withoffset=False*)

Returns a list of xrefs accessing the String.

The list contains tuples of the originating class and methods, where the class is represented as a *ClassAnalysis*, while the method is a *MethodAnalysis*.

is_overwritten()

Returns True if the string was overwritten :return:

set_value(*value*)

Overwrite the current value of the String with a new value. The original value is not lost and can still be retrieved using *get_orig_value()*.

Parameters value (*str*) – new string value

androguard.core.analysis.analysis.**is_ascii_obfuscation**(*vm*)

Tests if any class inside a DalvikVMObject uses ASCII Obfuscation (e.g. UTF-8 Chars in Classnames)

Parameters vm – *DalvikVMObject*

Returns True if ascii obfuscation otherwise False

androguard.core.analysis.auto module

class androguard.core.analysis.auto.**AndroAuto**(*settings*)

Bases: object

The main class which analyse automatically android apps by calling methods from a specific object

Automatic analysis requires two objects to be created:

1) a Logger, found at key *log* in the settings
2) an Analysis runner, found at key *my* in the settings

Both are passed to `AndroAuto` via a dictionary. The setting dict understands the following keys:

- *my*: The Analysis runner (required)
- *log*: The Logger
- *max_fetcher*: Maximum number of concurrent threads

`DefaultAndroLog` can be used as a baseclass for the Logger, while `DefaultAndroAnalysis` can be used a baseclass for the Analysis. There is also `DirectoryAndroAnalysis` which implements a *fetcher* which recursively reads a directory for files and can be used a baseclass as well.

example:

```
from androguard.core.analysis import auto

class AndroTest(auto.DirectoryAndroAnalysis):
    # This is the Test Runner
    def analysis_app(self, log, apkobj, dexobj, analysisobj):
        # Just print all objects to stdout
        print(log.id_file, log.filename, apkobj, dexobj, analysisobj)

settings = {
    # The directory `some/directory` should contain some APK files
    "my": AndroTest('some/directory'),
    # Use the default Logger
    "log": auto.DefaultAndroLog,
    # Use maximum of 2 threads
    "max_fetcher": 2,
}

aa = auto.AndroAuto(settings)
aa.go()
```

Parameters **`settings`** (`dict`) – the settings of the analysis

`dump`()

Dump the analysis

Calls *dump()* on the Analysis object

`dump_file`(*filename*)

Dump the analysis into a file

Calls *dump_file(filename)* on the Analysis object

`go`()

Launch the analysis.

this will start a total of *max_fetcher* threads.

class `androguard.core.analysis.auto.`**`DefaultAndroAnalysis`**

Bases: `object`

This class can be used as a template in order to analyse apps

The order of methods called in this class is the following:

- *fetcher()* is called to get files
- *filter_file()* is called to get the filetype
- *create_apk()* or *create_axml()* or *create_arsc()* and *create_dex()* or *create_dey()* depending on the filetype
- *analysis_apk()* or *analysis_axml()* or *analysis_arsc()* and *analysis_dex()* or *analysis_dey()* depending on the filetype
- *create_adex()* if at least one dex was found
- *analysis_app()* with all the gathered objects so far
- *finish()* is called in any case after the analysis

crash() can be called during analysis if any Exception happens.

analysis_adex(*log*, *adexobj*)
This method is called in order to know if the analysis must continue

Parameters

- **log** – an object which corresponds to a unique app
- **adexobj** (androguard.core.analysis.analysis.Analysis) – a Analysis object

Return type a boolean

analysis_apk(*log*, *apkobj*)
This method is called in order to know if the analysis must continue

Parameters

- **log** – an object which corresponds to a unique app
- **apkobj** (androguard.core.bytecodes.apk.APK) – a APK object

Returns True if a DEX file should be analyzed as well

Return type bool

analysis_app(*log*, *apkobj*, *dexobj*, *adexobj*)
This method is called if you wish to analyse the final app

Parameters

- **log** – an object which corresponds to a unique app
- **apkobj** (androguard.core.bytecodes.apk.APK) – a APK object
- **dexobj** (androguard.core.bytecodes.dvm.DalvikVMFormat) – a DalvikVMFormat object
- **adexobj** (androguard.core.analysis.analysis.Analysis) – a Analysis object

analysis_arsc(*log*, *arscobj*)
This method is called in order to know if the analysis must continue

Parameters

- **log** – an object which corresponds to a unique app

- **arscobj** (androguard.core.bytecodes.axml.ARSCParser) – a `ARSCParser` object

Returns True if the analysis should continue afterwards

Return type bool

analysis_axml(*log*, *axmlobj*)
This method is called in order to know if the analysis must continue

Parameters

- **log** – an object which corresponds to a unique app
- **axmlobj** (androguard.core.bytecodes.axml.AXMLPrinter) – a `AXMLPrinter` object

Returns True if the analysis should continue afterwards

Return type bool

analysis_dex(*log*, *dexobj*)
This method is called in order to know if the analysis must continue

Parameters

- **log** – an object which corresponds to a unique app
- **dexobj** (androguard.core.bytecodes.dvm.DalvikVMFormat) – a `DalvikVMFormat` object

Returns True if the analysis should continue with an analysis.Analysis

Return type bool

analysis_dey(*log*, *deyobj*)
This method is called in order to know if the analysis must continue

Parameters

- **log** – an object which corresponds to a unique app
- **deyobj** (androguard.core.bytecodes.dvm.DalvikOdexVMFormat) – a `DalvikOdexVMFormat` object

Returns True if the analysis should continue with an analysis.Analysis

Return type bool

crash(*log*, *why*)
This method is called if a crash happens

Parameters

- **log** – an object which corresponds to an unique app
- **why** – the exception

create_adex(*log*, *dexobj*)
This method is called in order to create an Analysis object

Parameters

- **log** – an object which corresponds to a unique app
- **dexobj** (androguard.core.bytecodes.dvm.DalvikVMFormat) – a `DalvikVMFormat` object

Rytpe a *Analysis* object

create_apk(*log*, *fileraw*)
This method is called in order to create a new APK object

Parameters

- **log** – an object which corresponds to a unique app
- **fileraw** – the raw apk (a string)

Return type an *APK* object

create_arsc(*log*, *fileraw*)
This method is called in order to create a new ARSC object

Parameters

- **log** – an object which corresponds to a unique app
- **fileraw** – the raw arsc (a string)

Return type an *ARSCParser* object

create_axml(*log*, *fileraw*)
This method is called in order to create a new AXML object

Parameters

- **log** – an object which corresponds to a unique app
- **fileraw** – the raw axml (a string)

Return type an *AXMLPrinter* object

create_dex(*log*, *dexraw*)
This method is called in order to create a DalvikVMFormat object

Parameters

- **log** – an object which corresponds to a unique app
- **dexraw** – the raw classes.dex (a string)

Return type a *DalvikVMFormat* object

create_dey(*log*, *dexraw*)
This method is called in order to create a DalvikOdexVMFormat object

Parameters

- **log** – an object which corresponds to a unique app
- **dexraw** – the raw odex file (a string)

Return type a *DalvikOdexVMFormat* object

dump()
This method is called to dump the result

dump_file(*filename*)
This method is called to dump the result in a file

Parameters **filename** – the filename to dump the result

fetcher(*q*)
This method is called to fetch a new app in order to analyse it. The queue must be fill with the following format: (filename, raw)

must return False if the queue is filled, thus all files are read.

Parameters **q** – the Queue to put new app

filter_file (*log*, *fileraw*)
This method is called in order to filer a specific app

Parameters

- **log** – an object which corresponds to a unique app
- **fileraw** (*bytes*) – the raw file as bytes

Return type a tuple with 2 elements, the return value (boolean) if it is necessary to continue the analysis and the file type

finish (*log*)
This method is called before the end of the analysis

Parameters **log** – an object which corresponds to an unique app

class androguard.core.analysis.auto.**DefaultAndroLog** (*id_file*, *filename*)
Bases: `object`

A base class for the Androguard Auto Logger.

The Logger contains two attributes of the analyzed File: `filename` and `id_file`, which is the Adler32 Checksum of the file.

The Logger can be extended to contain more attributes.

class androguard.core.analysis.auto.**DirectoryAndroAnalysis** (*directory*)
Bases: *androguard.core.analysis.auto.DefaultAndroAnalysis*

A simple class example to analyse a whole directory with many APKs in it

fetcher (*q*)
This method is called to fetch a new app in order to analyse it. The queue must be fill with the following format: (filename, raw)

must return False if the queue is filled, thus all files are read.

Parameters **q** – the Queue to put new app

Module contents

androguard.core.api_specific_resources package

Module contents

exception androguard.core.api_specific_resources.**APILevelNotFoundError**
Bases: `Exception`

androguard.core.api_specific_resources.**load_permission_mappings** (*apilevel*)
Load the API/Permission mapping for the requested API level. If the requetsed level was not found, None is returned.

Parameters **apilevel** – integer value of the API level, i.e. 24 for Android 7.0

Returns a dictionary of {MethodSignature: [List of Permissions]} androguard.core.api_specific_resources.**load_permissions**(*apilevel*, *permtype='permissions'*)

Load the Permissions for the given apilevel.

The permissions lists are generated using this tool: https://github.com/U039b/aosp_permissions_extraction

Has a fallback to select the maximum or minimal available API level. For example, if 28 is requested but only 26 is available, 26 is returned. If 5 is requested but 16 is available, 16 is returned.

If an API level is requested which is in between of two API levels we got, the lower level is returned. For example, if 5,6,7,10 is available and 8 is requested, 7 is returned instead.

Parameters

- **apilevel** – integer value of the API level
- **permtype** – either load permissions (`'permissions'`) or

permission groups (`'groups'`) :return: a dictionary of {Permission Name: {Permission info}

androguard.core.bytecodes package

The bytecodes modules are one very important core feature of Androguard. They contain parsers for APK, AXML, DEX, ODEX and DEY files as well for formats used inside these formats. These might be MUTF-8 for string encoding in DEX files as well as the widely used LEB128 encoding for numbers.

The most important modules might be *androguard.core.bytecodes.apk.APK* and *androguard.core.bytecodes.dvm.DalvikVMFormat*.

Submodules

androguard.core.bytecodes.apk module

class androguard.core.bytecodes.apk.**APK**(*filename*, *raw=False*, *magic_file=None*, *skip_analysis=False*, *testzip=False*)

Bases: `object`

property files

Returns a dictionary of filenames and detected magic type

Returns dictionary of files and their mime type

find_tags(*tag_name*, ***attribute_filter*)

Return a list of all the matched tags in all available xml

Parameters **tag** (*str*) – specify the tag name

find_tags_from_xml(*xml_name*, *tag_name*, ***attribute_filter*)

Return a list of all the matched tags in a specific xml w :param str xml_name: specify from which xml to pick the tag from :param str tag_name: specify the tag name

get_activities()

Return the android:name attribute of all activities

Return type a list of str

get_all_attribute_value(*tag_name*, *attribute*, *format_value=True*, ***attribute_filter*)

Yields all the attribute values in xml files which match with the tag name and the specific attribute

Parameters

- **tag_name** (*str*) – specify the tag name
- **attribute** (*str*) – specify the attribute
- **format_value** (*bool*) – specify if the value needs to be formatted with packagename

get_all_dex()

Return the raw data of all classes dex files

Return type a generator of bytes

get_android_manifest_axml()

Return the AXMLPrinter object which corresponds to the AndroidManifest.xml file

Return type *AXMLPrinter*

get_android_manifest_xml()

Return the parsed xml object which corresponds to the AndroidManifest.xml file

Return type Element

get_android_resources()

Return the *ARSCParser* object which corresponds to the resources.arsc file

Return type *ARSCParser*

get_androidversion_code()

Return the android version code

This information is read from the AndroidManifest.xml

Return type str

get_androidversion_name()

Return the android version name

This information is read from the AndroidManifest.xml

Return type str

get_app_icon(*max_dpi=65536*)

Return the first icon file name, which density is not greater than max_dpi, unless exact icon resolution is set in the manifest, in which case return the exact file.

This information is read from the AndroidManifest.xml

From https://developer.android.com/guide/practices/screens_support.html and https://developer.android.com/ndk/reference/group___configuration.html

- DEFAULT 0dpi
- ldpi (low) 120dpi
- mdpi (medium) 160dpi
- TV 213dpi
- hdpi (high) 240dpi
- xhdpi (extra-high) 320dpi
- xxhdpi (extra-extra-high) 480dpi
- xxxhdpi (extra-extra-extra-high) 640dpi
- anydpi 65534dpi (0xFFFE)
- nodpi 65535dpi (0xFFFF)

There is a difference between nodpi and anydpi: nodpi will be used if no other density is specified. Or the density does not match. nodpi is the fallback for everything else. If there is a resource that matches the DPI, this is used. anydpi is also valid for all densities but in this case, anydpi will overrule all other files! Therefore anydpi is usually used with vector graphics and with constraints on the API level. For example adaptive icons are usually marked as anydpi.

When it comes now to selecting an icon, there is the following flow:

1. is there an anydpi icon?
2. is there an icon for the dpi of the device?
3. is there a nodpi icon?
4. (only on very old devices) is there a icon with dpi 0 (the default)

For more information read here: https://stackoverflow.com/a/34370735/446140

Return type `str`

`get_app_name``()`

Return the appname of the APK

This name is read from the AndroidManifest.xml using the application android:label. If no label exists, the android:label of the main activity is used.

If there is also no main activity label, an empty string is returned.

Return type `str`

`get_attribute_value``(`*tag_name*, *attribute*, *format_value=False*, ***attribute_filter*`)`

Return the attribute value in xml files which matches the tag name and the specific attribute

Parameters

- **`tag_name`** (*`str`*) – specify the tag name
- **`attribute`** (*`str`*) – specify the attribute
- **`format_value`** (*`bool`*) – specify if the value needs to be formatted with packagename

`get_certificate``(`*filename*`)`

Return a X.509 certificate object by giving the name in the apk file

Parameters **`filename`** – filename of the signature file in the APK

Returns a `Certificate` certificate

`get_certificate_der``(`*filename*`)`

Return the DER coded X.509 certificate from the signature file.

Parameters **`filename`** – Signature filename in APK

Returns DER coded X.509 certificate as binary

`get_certificates``()`

Return a list of unique `asn1crypto.x509.Certificate` which are found in v1, v2 and v3 signing Note that we simply extract all certificates regardless of the signer. Therefore this is just a list of all certificates found in all signers.

`get_certificates_der_v2``()`

Return a list of DER coded X.509 certificates from the v3 signature block

`get_certificates_der_v3``()`

Return a list of DER coded X.509 certificates from the v3 signature block

get_certificates_v1()

Return a list of `asn1crypto.x509.Certificate` which are found in the META-INF folder (v1 signing). Note that we simply extract all certificates regardless of the signer. Therefore this is just a list of all certificates found in all signers.

get_certificates_v2()

Return a list of `asn1crypto.x509.Certificate` which are found in the v2 signing block. Note that we simply extract all certificates regardless of the signer. Therefore this is just a list of all certificates found in all signers.

get_certificates_v3()

Return a list of `asn1crypto.x509.Certificate` which are found in the v3 signing block. Note that we simply extract all certificates regardless of the signer. Therefore this is just a list of all certificates found in all signers.

get_declared_permissions()

Returns list of the declared permissions.

Return type list of strings

get_declared_permissions_details()

Returns declared permissions with the details.

Return type dict

get_details_permissions()

Return permissions with details.

THis can only return details about the permission, if the permission is defined in the AOSP.

Return type dict of {permission: [protectionLevel, label, description]}

get_dex()

Return the raw data of the classes dex file

This will give you the data of the file called *classes.dex* inside the APK. If the APK has multiple DEX files, you need to use `get_all_dex()`.

Return type bytes

get_dex_names()

Return the names of all DEX files found in the APK. This method only accounts for "offical" dex files, i.e. all files in the root directory of the APK named classes.dex or classes[0-9]+.dex

Return type a list of str

get_effective_target_sdk_version()

Return the effective targetSdkVersion, always returns int > 0.

If the targetSdkVersion is not set, it defaults to 1. This is set based on defaults as defined in: https://developer.android.com/guide/topics/manifest/uses-sdk-element.html

Return type int

get_element(*tag_name*, *attribute*, ***attribute_filter*)

Deprecated since version 3.3.5: use `get_attribute_value()` instead

Return element in xml files which match with the tag name and the specific attribute

Parameters

- **tag_name** (*str*) – specify the tag name
- **attribute** (*str*) – specify the attribute

Return type str

get_elements(*tag_name*, *attribute*, *with_namespace=True*)

Deprecated since version 3.3.5: use `get_all_attribute_value()` instead

Return elements in xml files which match with the tag name and the specific attribute

Parameters

- **tag_name** (`str`) – a string which specify the tag name
- **attribute** (`str`) – a string which specify the attribute

get_features()

Return a list of all android:names found for the tag uses-feature in the AndroidManifest.xml

Returns list

get_file(*filename*)

Return the raw data of the specified filename inside the APK

Return type bytes

get_filename()

Return the filename of the APK

Return type `str`

get_files()

Return the file names inside the APK.

Return type a list of `str`

get_files_crc32()

Calculates and returns a dictionary of filenames and CRC32

Returns dict of filename: CRC32

get_files_information()

Return the files inside the APK with their associated types and crc32

Return type str, str, int

get_files_types()

Return the files inside the APK with their associated types (by using python-magic)

At the same time, the CRC32 are calculated for the files.

Return type a dictionnary

get_intent_filters(*itemtype*, *name*)

Find intent filters for a given item and name.

Intent filter are attached to activities, services or receivers. You can search for the intent filters of such items and get a dictionary of all attached actions and intent categories.

Parameters

- **itemtype** – the type of parent item to look for, e.g. *activity*, *service* or *receiver*
- **name** – the *android:name* of the parent item, e.g. activity name

Returns a dictionary with the keys *action* and *category* containing the *android:name* of those items

get_libraries()

Return the android:name attributes for libraries

Return type list

get_main_activities()

Return names of the main activities

These values are read from the AndroidManifest.xml

Return type a set of str

get_main_activity()

Return the name of the main activity

This value is read from the AndroidManifest.xml

Return type str

get_max_sdk_version()

Return the android:maxSdkVersion attribute

Return type string

get_min_sdk_version()

Return the android:minSdkVersion attribute

Return type string

get_package()

Return the name of the package

This information is read from the AndroidManifest.xml

Return type `str`

get_permissions()

Return permissions names declared in the AndroidManifest.xml.

It is possible that permissions are returned multiple times, as this function does not filter the permissions, i.e. it shows you exactly what was defined in the AndroidManifest.xml.

Implied permissions, which are granted automatically, are not returned here. Use *get_uses_implied_permission_list()* if you need a list of implied permissions.

Returns A list of permissions

Return type list

get_providers()

Return the android:name attribute of all providers

Return type a list of string

get_public_keys_der_v2()

Return a list of DER coded X.509 public keys from the v3 signature block

get_public_keys_der_v3()

Return a list of DER coded X.509 public keys from the v3 signature block

get_public_keys_v2()

Return a list of `asn1crypto.keys.PublicKeyInfo` which are found in the v2 signing block.

get_public_keys_v3()

Return a list of `asn1crypto.keys.PublicKeyInfo` which are found in the v3 signing block.

get_raw()

Return raw bytes of the APK

Return type bytes

get_receivers()
Return the android:name attribute of all receivers

Return type a list of string

get_requested_aosp_permissions()
Returns requested permissions declared within AOSP project.

This includes several other permissions as well, which are in the platform apps.

Return type list of str

get_requested_aosp_permissions_details()
Returns requested aosp permissions with details.

Return type dictionary

get_requested_permissions()
Deprecated since version 3.1.0: use *get_permissions()* instead.

Returns all requested permissions.

It has the same result as *get_permissions()* and might be removed in the future

Return type list of str

get_requested_third_party_permissions()
Returns list of requested permissions not declared within AOSP project.

Return type list of strings

get_res_value(*name*)
Return the literal value with a resource id

Return type str

get_services()
Return the android:name attribute of all services

Return type a list of str

get_signature()
Return the data of the first signature file found (v1 Signature / JAR Signature)

Return type First signature name or None if not signed

get_signature_name()
Return the name of the first signature file found.

get_signature_names()
Return a list of the signature file names (v1 Signature / JAR Signature)

Return type List of filenames matching a Signature

get_signatures()
Return a list of the data of the signature files. Only v1 / JAR Signing.

Return type list of bytes

get_target_sdk_version()
Return the android:targetSdkVersion attribute

Return type string

get_uses_implied_permission_list()
Return all permissions implied by the target SDK or other permissions.

Return type list of string

get_value_from_tag (*tag*, *attribute*)

Return the value of the android prefixed attribute in a specific tag.

This function will always try to get the attribute with a android: prefix first, and will try to return the attribute without the prefix, if the attribute could not be found. This is useful for some broken AndroidManifest.xml, where no android namespace is set, but could also indicate malicious activity (i.e. wrongly repackaged files). A warning is printed if the attribute is found without a namespace prefix.

If you require to get the exact result you need to query the tag directly:

example::

```
>>> from lxml.etree import Element
>>> tag = Element('bar', nsmap={'android': 'http://schemas.android.com/
↪apk/res/android'})
>>> tag.set('{http://schemas.android.com/apk/res/android}foobar', 'barfoo
↪')
>>> tag.set('name', 'baz')
# Assume that `a` is some APK object
>>> a.get_value_from_tag(tag, 'name')
'baz'
>>> tag.get('name')
'baz'
>>> tag.get('foobar')
None
>>> a.get_value_from_tag(tag, 'foobar')
'barfoo'
```

Parameters

- **tag** (*lxml.etree.Element*) – specify the tag element
- **attribute** (*str*) – specify the attribute name

Returns the attribute's value, or None if the attribute is not present

is_androidtv ()

Checks if this application does not require a touchscreen, as this is the rule to get into the TV section of the Play Store See: https://developer.android.com/training/tv/start/start.html for more information.

Returns True if 'android.hardware.touchscreen' is not required, False otherwise

is_leanback ()

Checks if this application is build for TV (Leanback support) by checkin if it uses the feature 'android.software.leanback'

Returns True if leanback feature is used, false otherwise

is_multidex ()

Test if the APK has multiple DEX files

Returns True if multiple dex found, otherwise False

is_signed ()

Returns true if either a v1 or v2 (or both) signature was found.

is_signed_v1 ()

Returns true if a v1 / JAR signature was found.

Returning *True* does not mean that the file is properly signed! It just says that there is a signature file which needs to be validated.

is_signed_v2()

Returns true of a v2 / APK signature was found.

Returning *True* does not mean that the file is properly signed! It just says that there is a signature file which needs to be validated.

is_signed_v3()

Returns true of a v3 / APK signature was found.

Returning *True* does not mean that the file is properly signed! It just says that there is a signature file which needs to be validated.

is_tag_matched(*tag*, ***attribute_filter*)

Return true if the attributes matches in attribute filter.

An attribute filter is a dictionary containing: {attribute_name: value}. This function will return True if and only if all attributes have the same value. This function allows to set the dictionary via kwargs, thus you can filter like this:

example:: a.is_tag_matched(tag, name=”foobar”, other=”barfoo”)

This function uses a fallback for attribute searching. It will by default use the namespace variant but fall back to the non-namespace variant. Thus specifiying `{"name": "foobar"}` will match on `<bla name="foobar" \>` as well as on `<bla android:name="foobar" \>`.

Parameters

- **tag** (*lxml.etree.Element*) – specify the tag element
- **attribute_filter** – specify the attribute filter as dictionary

is_valid_APK()

Return true if the APK is valid, false otherwise. An APK is seen as valid, if the AndroidManifest.xml could be successful parsed. This does not mean that the APK has a valid signature nor that the APK can be installed on an Android system.

Return type boolean

is_wearable()

Checks if this application is build for wearables by checking if it uses the feature ‘android.hardware.type.watch’ See: https://developer.android.com/training/wearables/apps/creating.html for more information.

Not every app is setting this feature (not even the example Google provides), so it might be wise to not 100% rely on this feature.

Returns True if wearable, False otherwise

new_zip(*filename*, *deleted_files=None*, *new_files={}*)

Create a new zip file

Parameters

- **filename** (*string*) – the output filename of the zip
- **deleted_files** (*None or a string*) – a regex pattern to remove specific file
- **new_files** (*a dictionnary (key:filename, value:content of the file)*) – a dictionnary of new files

parse_signatures_or_digests(*digest_bytes*)

Parse digests

parse_v2_signing_block()
Parse the V2 signing block and extract all features

parse_v2_v3_signature()

parse_v3_signing_block()
Parse the V2 signing block and extract all features

read_uint32_le(*io_stream*)

show()

class androguard.core.bytecodes.apk.**APKV2SignedData**
Bases: object

This class holds all data associated with an APK V3 SigningBlock signed data. source : https://source.android.com/security/apksigning/v2.html

class androguard.core.bytecodes.apk.**APKV2Signer**
Bases: object

This class holds all data associated with an APK V2 SigningBlock signer. source : https://source.android.com/security/apksigning/v2.html

class androguard.core.bytecodes.apk.**APKV3SignedData**
Bases: *androguard.core.bytecodes.apk.APKV2SignedData*

This class holds all data associated with an APK V3 SigningBlock signed data. source : https://source.android.com/security/apksigning/v3.html

class androguard.core.bytecodes.apk.**APKV3Signer**
Bases: *androguard.core.bytecodes.apk.APKV2Signer*

This class holds all data associated with an APK V3 SigningBlock signer. source : https://source.android.com/security/apksigning/v3.html

exception androguard.core.bytecodes.apk.**BrokenAPKError**
Bases: *androguard.core.bytecodes.apk.Error*

exception androguard.core.bytecodes.apk.**Error**
Bases: Exception

Base class for exceptions in this module.

exception androguard.core.bytecodes.apk.**FileNotPresent**
Bases: *androguard.core.bytecodes.apk.Error*

androguard.core.bytecodes.apk.**ensure_final_value**(*packageName*, *arsc*, *value*)
Ensure incoming value is always the value, not the resid

androguard will sometimes return the Android “resId” aka Resource ID instead of the actual value. This checks whether the value is actually a resId, then performs the Android Resource lookup as needed.

androguard.core.bytecodes.apk.**get_apkid**(*apkfile*)
Read (appid, versionCode, versionName) from an APK

This first tries to do quick binary XML parsing to just get the values that are needed. It will fallback to full androguard parsing, which is slow, if it can’t find the versionName value or versionName is set to a Android String Resource (e.g. an integer hex value that starts with @).

androguard.core.bytecodes.apk.**parse_lxml_dom**(*tree*)

androguard.core.bytecodes.apk.**show_Certificate**(*cert*, *short=False*)
Print Fingerprints, Issuer and Subject of an X509 Certificate.

Parameters

- **cert** (asn1crypto.x509.Certificate) – X509 Certificate to print
- **short** (*Boolean*) – Print in shortform for DN (Default: False)

androguard.core.bytecodes.dvm module

class androguard.core.bytecodes.dvm.**AnnotationElement**(*buff*, *cm*)

Bases: object

This class can parse an annotation_element of a dex file

Parameters

- **buff** (*Buff object*) – a string which represents a Buff object of the annotation_element
- **cm** (*ClassManager*) – a ClassManager object

get_length()

get_name_idx()

Return the element name, represented as an index into the string_ids section

Return type int

get_obj()

get_raw()

get_value()

Return the element value (EncodedValue)

Return type a *EncodedValue* object

show()

class androguard.core.bytecodes.dvm.**AnnotationItem**(*buff*, *cm*)

Bases: object

This class can parse an annotation_item of a dex file

Parameters

- **buff** (*Buff object*) – a string which represents a Buff object of the annotation_item
- **cm** (*ClassManager*) – a ClassManager object

get_annotation()

Return the encoded annotation contents

Return type a *EncodedAnnotation* object

get_length()

get_obj()

get_off()

get_raw()

get_visibility()

Return the intended visibility of this annotation

Return type int

set_off(*off*)

show()

class androguard.core.bytecodes.dvm.**AnnotationOffItem**(*buff*, *cm*)

Bases: `object`

This class can parse an annotation_off_item of a dex file

Parameters

- **buff** (*Buff object*) – a string which represents a Buff object of the annotation_off_item
- **cm** (*ClassManager*) – a ClassManager object

get_annotation_item()

get_annotation_off()

get_length()

get_obj()

get_raw()

show()

class androguard.core.bytecodes.dvm.**AnnotationSetItem**(*buff*, *cm*)

Bases: `object`

This class can parse an annotation_set_item of a dex file

Parameters

- **buff** – a string which represents a Buff object of the annotation_set_item
- **cm** (*ClassManager*) – a ClassManager object

get_annotation_off_item()

Return the offset from the start of the file to an annotation

Return type a list of *AnnotationOffItem*

get_length()

get_obj()

get_off()

get_raw()

set_off(*off*)

show()

class androguard.core.bytecodes.dvm.**AnnotationSetRefItem**(*buff*, *cm*)

Bases: `object`

This class can parse an annotation_set_ref_item of a dex file

Parameters

- **buff** (*Buff object*) – a string which represents a Buff object of the annotation_set_ref_item
- **cm** (*ClassManager*) – a ClassManager object

get_annotations_off()

Return the offset from the start of the file to the referenced annotation set or 0 if there are no annotations for this element.

Return type int

get_obj()

get_raw()

show()

class androguard.core.bytecodes.dvm.**AnnotationSetRefList**(*buff*, *cm*)

Bases: object

This class can parse an annotation_set_ref_list_item of a dex file

Parameters

- **buff** (*Buff object*) – a string which represents a Buff object of the annotation_set_ref_list_item
- **cm** (*ClassManager*) – a ClassManager object

get_length()

get_list()

Return elements of the list

Return type *AnnotationSetRefItem*

get_obj()

get_off()

get_raw()

set_off(*off*)

show()

class androguard.core.bytecodes.dvm.**AnnotationsDirectoryItem**(*buff*, *cm*)

Bases: object

This class can parse an annotations_directory_item of a dex file

Parameters

- **buff** (*Buff object*) – a string which represents a Buff object of the annotations_directory_item
- **cm** (*ClassManager*) – a ClassManager object

get_annotated_fields_size()

Return the count of fields annotated by this item

Return type int

get_annotated_methods_size()

Return the count of methods annotated by this item

Return type int

get_annotated_parameters_size()

Return the count of method parameter lists annotated by this item

Return type int

get_annotation_set_item()

get_class_annotations_off()
Return the offset from the start of the file to the annotations made directly on the class, or 0 if the class has no direct annotations

Return type int

get_field_annotations()
Return the list of associated field annotations

Return type a list of *FieldAnnotation*

get_length()

get_method_annotations()
Return the list of associated method annotations

Return type a list of *MethodAnnotation*

get_obj()

get_off()

get_parameter_annotations()
Return the list of associated method parameter annotations

Return type a list of *ParameterAnnotation*

get_raw()

set_off(*off*)

show()

class androguard.core.bytecodes.dvm.**ClassDataItem**(*buff*, *cm*)
Bases: object

This class can parse a class_data_item of a dex file

Parameters

- **buff** (*Buff object*) – a string which represents a Buff object of the class_data_item
- **cm** (*ClassManager*) – a ClassManager object

get_direct_methods()
Return the defined direct (any of static, private, or constructor) methods, represented as a sequence of encoded elements

Return type a list of *EncodedMethod* objects

get_direct_methods_size()
Return the number of direct methods defined in this item

Return type int

get_fields()
Return static and instance fields

Return type a list of *EncodedField* objects

get_instance_fields()
Return the defined instance fields, represented as a sequence of encoded elements

Return type a list of *EncodedField* objects

get_instance_fields_size()
Return the number of instance fields defined in this item

Return type int

get_length()

get_methods()
Return direct and virtual methods

Return type a list of *EncodedMethod* objects

get_obj()

get_off()

get_raw()

get_static_fields()
Return the defined static fields, represented as a sequence of encoded elements

Return type a list of *EncodedField* objects

get_static_fields_size()
Return the number of static fields defined in this item

Return type int

get_virtual_methods()
Return the defined virtual (none of static, private, or constructor) methods, represented as a sequence of encoded elements

Return type a list of *EncodedMethod* objects

get_virtual_methods_size()
Return the number of virtual methods defined in this item

Return type int

set_off(*off*)

set_static_fields(*value*)

show()

class androguard.core.bytecodes.dvm.**ClassDefItem**(*buff*, *cm*)
Bases: object

This class can parse a class_def_item of a dex file

Parameters

- **buff** (*Buff object*) – a string which represents a Buff object of the class_def_item
- **cm** (*ClassManager*) – a ClassManager object

get_access_flags()
Return the access flags for the class (public, final, etc.)

Return type int

get_access_flags_string()
Return the access flags string of the class

Return type str

get_annotations()

get_annotations_off()
Return the offset from the start of the file to the annotations structure for this class, or 0 if there are no annotations on this class.

Return type int

get_ast()

get_class_data()
Return the associated class_data_item

Return type a *ClassDataItem* object

get_class_data_off()
Return the offset from the start of the file to the associated class data for this item, or 0 if there is no class data for this class

Return type int

get_class_idx()
Return the index into the type_ids list for this class

Return type int

get_fields()
Return all fields of this class

Return type a list of *EncodedField* objects

get_interfaces()
Return the names of the interfaces

Return type List[*MUTF8String*]

get_interfaces_off()
Return the offset from the start of the file to the list of interfaces, or 0 if there are none

Return type int

get_length()

get_methods()
Return all methods of this class

Return type a list of *EncodedMethod* objects

get_name()
Return the name of this class

Return type *MUTF8String*

get_obj()

get_raw()

get_source()

get_source_ext()

get_source_file_idx()
Return the index into the string_ids list for the name of the file containing the original source for (at least most of) this class, or the special value NO_INDEX to represent a lack of this information

Return type int

get_static_values_off()
Return the offset from the start of the file to the list of initial values for static fields, or 0 if there are none (and all static fields are to be initialized with 0 or null)

Return type int

get_superclass_idx()

Return the index into the type_ids list for the superclass

Return type int

get_superclassname()

Return the name of the super class

Return type *MUTF8String*

reload()

set_name(*value*)

show()

source()

Return the source code of the entire class

Return type string

class androguard.core.bytecodes.dvm.**ClassHDefItem**(*size*, *buff*, *cm*)

Bases: object

This class can parse a list of class_def_item of a dex file

Parameters

- **buff** (*Buff object*) – a string which represents a Buff object of the list of class_def_item
- **cm** (*ClassManager*) – a ClassManager object

get_class_idx(*idx*)

get_length()

get_method(*name_class*, *name_method*)

get_names()

get_obj()

get_off()

get_raw()

set_off(*off*)

show()

class androguard.core.bytecodes.dvm.**ClassManager**(*vm*)

Bases: object

This class is used to access to all elements (strings, type, proto ...) of the dex format based on their offset or index.

add_type_item(*type_item*, *c_item*, *item*)

get_all_engine()

Deprecated since version 3.3.5: do not use this function anymore!

get_annotation_item(*off*)

get_annotation_off_item(*off*)

get_annotation_set_item(*off*)

get_annotations_directory_item(*off*)

get_ascii_string(*s*)

get_class_data_item(*off*)

get_code(*idx*)

get_debug_off(*off*)

get_encoded_array_item(*off*)

get_engine()
Deprecated since version 3.3.5: do not use this function anymore!

get_field(*idx*)

get_field_ref(*idx*)

get_item_by_offset(*offset*)

get_lazy_analysis()
Deprecated since version 3.3.5: do not use this function anymore!

get_method(*idx*)

get_method_ref(*idx*)

get_next_offset_item(*idx*)

get_obj_by_offset(*offset*)
Returnes a object from as given offset inside the DEX file

get_odex_format()
Returns True if the underlying VM is ODEX

get_proto(*idx*)

get_raw_string(*idx*)
Return the (unprocessed) string from the string table at index *idx*.

Parameters **idx** (`int`) – the index in the string section

get_string(*idx*)
Return a string from the string table at index *idx*

If string is hooked, the hooked string is returned.

Parameters **idx** (`int`) – index in the string section

get_string_by_offset(*offset*)

get_type(*idx*)
Return the resolved type name based on the index

This returns the string associated with the type.

Parameters **idx** (`int`) –

Returns the type name

Return type str

get_type_list(*off*)

get_type_ref(*idx*)
Returns the string reference ID for a given type ID.

This method is similar to `get_type()` but does not resolve the string but returns the ID into the string section.

If the type IDX is not found, -1 is returned.

property packer

set_decompiler(*decompiler*)

set_hook_class_name(*class_def*, *value*)

set_hook_field_name(*encoded_field*, *value*)

set_hook_method_name(*encoded_method*, *value*)

set_hook_string(*idx*, *value*)

class androguard.core.bytecodes.dvm.**CodeItem**(*size*, *buff*, *cm*)
Bases: object

get_code(*off*)

get_length()

get_obj()

get_off()

get_raw()

set_off(*off*)

show()

class androguard.core.bytecodes.dvm.**DBGBytecode**(*cm*, *op_value*)
Bases: object

add(*value*, *ttype*)

get_obj()

get_op_value()

get_raw()

get_value()

show()

class androguard.core.bytecodes.dvm.**DCode**(*class_manager*, *offset*, *size*, *buff*)
Bases: object

This class represents the instructions of a method

Parameters

- **class_manager** (*ClassManager* object) – the ClassManager
- **offset** (*int*) – the offset of the buffer
- **size** (*int*) – the total size of the buffer
- **buff** (*string*) – a raw buffer where are the instructions

add_inote(*msg*, *idx*, *off=None*)
Add a message to a specific instruction by using (default) the index of the address if specified

Parameters

- **msg** (*string*) – the message
- **idx** (*int*) – index of the instruction (the position in the list of the instruction)

- **off** (*int*) – address of the instruction

get_ins_off(*off*)

Get a particular instruction by using the address

Parameters **off** (*int*) – address of the instruction

Return type an *Instruction* object

get_insn()

Get the insn buffer

Return type bytes

get_instruction(*idx*, *off=None*)

Get a particular instruction by using (default) the index of the address if specified

Parameters

- **idx** (*int*) – index of the instruction (the position in the list of the instruction)
- **off** (*int*) – address of the instruction

Return type an *Instruction* object

get_instructions()

Get the instructions

Return type a generator of each *Instruction* (or a cached list of instructions if you have setup instructions)

get_length()

Return the length of this object

Return type int

get_raw()

Return the raw buffer of this object

Return type bytearray

is_cached_instructions()

off_to_pos(*off*)

Get the position of an instruction by using the address

Parameters **off** (*int*) – address of the instruction

Return type int

set_idx(*idx*)

Set the start address of the buffer

Parameters **idx** (*int*) – the index

set_insn(*insn*)

Set a new raw buffer to disassemble

Parameters **insn** (*bytes*) – the buffer

set_instructions(*instructions*)

Set the instructions

Parameters **instructions** (a list of *Instruction*) – the list of instructions

show()

Display (with a pretty print) this object

class androguard.core.bytecodes.dvm.**DalvikCode**(*buff*, *cm*)

Bases: `object`

This class represents the instructions of a method

Parameters

- **buff** (`BuffHandle`) – a raw buffer where are the instructions
- **cm** (*ClassManager* object) – the ClassManager

add_inote(*msg*, *idx*, *off=None*)

Add a message to a specific instruction by using (default) the index of the address if specified

Parameters

- **msg** (*string*) – the message
- **idx** (*int*) – index of the instruction (the position in the list of the instruction)
- **off** (*int*) – address of the instruction

get_bc()

Return the associated code object

Return type *DCode*

get_debug()

Return the associated debug object

Return type *DebugInfoItem*

get_debug_info_off()

Get the offset from the start of the file to the debug info (line numbers + local variable info) sequence for this code, or 0 if there simply is no information

Return type int

get_handlers()

Get the bytes representing a list of lists of catch types and associated handler addresses.

Return type *EncodedCatchHandlerList*

get_ins_size()

Get the number of words of incoming arguments to the method that this code is for

Return type int

get_insns_size()

Get the size of the instructions list, in 16-bit code units

Return type int

get_instruction(*idx*, *off=None*)

get_length()

get_obj()

get_off()

get_outs_size()

Get the number of words of outgoing argument space required by this code for method invocation

Return type int

get_raw()

Get the reconstructed code as bytearray

Return type bytearray

get_registers_size()

Get the number of registers used by this code

Return type int

get_size()

get_tries()

Get the array indicating where in the code exceptions are caught and how to handle them

Return type a list of `TryItem` objects

get_tries_size()

Get the number of `TryItem` for this instance

Return type int

set_idx(*idx*)

set_off(*off*)

show()

class androguard.core.bytecodes.dvm.**DalvikOdexVMFormat**(*buff*, *decompiler=None*, *config=None*, *using_api=None*)

Bases: `androguard.core.bytecodes.dvm.DalvikVMFormat`

This class can parse an odex file

Parameters

- **buff** (`string`) – a string which represents the odex file
- **decompiler** (`object`) – associate a decompiler object to display the java source code

Example DalvikOdexVMFormat(read("classes.odex"))

get_buff()

Return the whole buffer

Return type bytearray

get_dependencies()

Return the odex dependencies object

Return type an OdexDependencies object

get_format_type()

Return the type

Return type a string

save()

Do not use !

class androguard.core.bytecodes.dvm.**DalvikPacker**(*endian_tag*)

Bases: `object`

Generic Packer class to unpack bytes based on different endianness

class androguard.core.bytecodes.dvm.**DalvikVMFormat**(*buff*, *decompiler=None*, *config=None*, *using_api=None*)

Bases: `androguard.core.bytecode.BuffHandle`

This class can parse a classes.dex file of an Android application (APK).

Parameters

- **buff** (*bytes*) – a string which represents the classes.dex file
- **decompiler** (*object*) – associate a decompiler object to display the java source code

example:

```
d = DalvikVMFormat( read("classes.dex") )
```

create_python_export()
Export classes/methods/fields' names in the python namespace

disassemble(*offset*, *size*)
Disassembles a given offset in the DEX file

Parameters

- **offset** (*int*) – offset to disassemble in the file (from the beginning of the file)
- **size** –

fix_checksums(*buff*)
Fix a dex format buffer by setting all checksums

Return type string

get_BRANCH_DVM_OPCODES()
Deprecated since version 3.4.0: Will be removed!

get_all_fields()
Return a list of field items

Return type a list of *FieldIdItem* objects

get_api_version()
This method returns api version that should be used for loading api specific resources.

Return type int

get_class(*name*)
Return a specific class

Parameters **name** – the name of the class

Return type a *ClassDefItem* object

get_class_manager()
This function returns a ClassManager object which allow you to get access to all index references (strings, methods, fields, ...)

Return type *ClassManager* object

get_classes()
Return all classes

Return type a list of *ClassDefItem* objects

get_classes_def_item()
This function returns the class def item

Return type *ClassHDefItem* object

get_classes_names(*update=False*)
Return the names of classes

Parameters **update** – True indicates to recompute the list. Maybe needed after using a MyClass.set_name().

Return type a list of string

get_cm_field(*idx*)
Get a specific field by using an index

Parameters **idx** (*int*) – index of the field

get_cm_method(*idx*)
Get a specific method by using an index

Parameters **idx** (*int*) – index of the method

get_cm_string(*idx*)
Get a specific string by using an index

Parameters **idx** (*int*) – index of the string

get_cm_type(*idx*)
Get a specific type by using an index

Parameters **idx** (*int*) – index of the type

get_codes_item()
This function returns the code item

Return type *CodeItem* object

get_debug_info_item()
This function returns the debug info item

Return type *DebugInfoItem* object

get_determineException()
Deprecated since version 3.4.0: Will be removed!

get_determineNext()
Deprecated since version 3.4.0: Will be removed!

get_field(*name*)
Return a list all fields which corresponds to the regexp

Parameters **name** – the name of the field (a python regexp)

Return type a list with all *EncodedField* objects

get_field_descriptor(*class_name*, *field_name*, *descriptor*)
Return the specific field

Parameters

- **class_name** (*string*) – the class name of the field
- **field_name** (*string*) – the name of the field
- **descriptor** (*string*) – the descriptor of the field

Return type None or a *EncodedField* object

get_fields()
Return all field objects

Return type a list of *EncodedField* objects

get_fields_class(*class_name*)
Return all fields of a specific class

Parameters **class_name** (*string*) – the class name

Return type a list with *EncodedField* objects

get_fields_id_item()
This function returns the field id item

Return type *FieldHIdItem* object

get_format()
Deprecated since version 3.4.0: Will be removed!

get_format_type()
Return the type

Return type a string

get_header_item()
This function returns the header item

Return type *HeaderItem* object

get_len_methods()
Return the number of methods

Return type int

get_method(*name*)
Return a list all methods which corresponds to the regexp

Parameters **name** – the name of the method (a python regexp)

Return type a list with all *EncodedMethod* objects

get_method_by_idx(*idx*)
Return a specific method by using an index :param idx: the index of the method :type idx: int

Return type None or an *EncodedMethod* object

get_method_descriptor(*class_name*, *method_name*, *descriptor*)
Return the specific method

Parameters

- **class_name** (*string*) – the class name of the method
- **method_name** (*string*) – the name of the method
- **descriptor** (*string*) – the descriptor of the method

Return type None or a *EncodedMethod* object

get_methods()
Return all method objects

Return type a list of *EncodedMethod* objects

get_methods_class(*class_name*)
Return all methods of a specific class

Parameters **class_name** (*string*) – the class name

Return type a list with *EncodedMethod* objects

get_methods_descriptor(*class_name*, *method_name*)
Return the specific methods of the class

Parameters

- **class_name** (*string*) – the class name of the method
- **method_name** (*string*) – the name of the method

Return type None or a `EncodedMethod` object

get_methods_id_item()
This function returns the method id item

Return type `MethodHIdItem` object

get_regex_strings(*regular_expressions*)
Return all target strings matched the regex

Parameters **regular_expressions** (*string*) – the python regex

Return type a list of strings matching the regex expression

get_string_data_item()
This function returns the string data item

Return type `StringDataItem` object

get_strings()
Return all strings

The strings will have escaped surrogates, if only a single high or low surrogate is found. Complete surrogates are put together into the representing 32bit character.

Return type a list with all strings used in the format (types, names ...)

get_vmanalysis()
Deprecated since version 3.1.0: The `Analysis` is not loaded anymore into `DalvikVMFormat` in order to avoid cyclic dependencies. `Analysis` extends now `DalvikVMFormat`. This Method does nothing anymore!

The Analysis Object should contain all the information required, inclduing the DalvikVMFormats.

list_classes_hierarchy()
Get a tree structure of the classes. The parent is always the superclass.

You can use pprint.pprint to print the dictionary in a pretty way.

Returns a dict with all the classnames

Return type dict

print_classes_hierarchy()
Deprecated since version 3.4.0: Will be removed!

save()
Return the dex (with the modifications) into raw format (fix checksums) (beta: do not use !)

Return type string

set_decompiler(*decompiler*)

set_vmanalysis(*analysis*)
Deprecated since version 3.1.0: The `Analysis` is not loaded anymore into `DalvikVMFormat` in order to avoid cyclic dependencies. `Analysis` extends now `DalvikVMFormat`. This Method does nothing anymore!

The Analysis Object should contain all the information required, inclduing the DalvikVMFormats.

show()
Show the all information in the object

property version
Returns the version number of the DEX Format

class androguard.core.bytecodes.dvm.**DebugInfoItem**(*buff*, *cm*)
Bases: object

get_bytecodes()

get_line_start()

get_off()

get_parameter_names()

get_parameters_size()

get_raw()

get_translated_parameter_names()

show()

class androguard.core.bytecodes.dvm.**DebugInfoItemEmpty**(*buff*, *cm*)
Bases: object

get_length()

get_obj()

get_off()

get_raw()

reload()

set_off(*off*)

show()

class androguard.core.bytecodes.dvm.**EncodedAnnotation**(*buff*, *cm*)
Bases: object

This class can parse an encoded_annotation of a dex file

Parameters

- **buff** (*Buff object*) – a string which represents a Buff object of the encoded_annotation
- **cm** (*ClassManager*) – a ClassManager object

get_elements()
Return the elements of the annotation, represented directly in-line (not as offsets)

Return type a list of *AnnotationElement* objects

get_length()

get_obj()

get_raw()

get_size()

Return the number of name-value mappings in this annotation

:rtype:int

get_type_idx()

Return the type of the annotation. This must be a class (not array or primitive) type

Return type int

show()

class androguard.core.bytecodes.dvm.**EncodedArray**(*buff*, *cm*)

Bases: `object`

This class can parse an encoded_array of a dex file

Parameters

- **buff** (*Buff object*) – a string which represents a Buff object of the encoded_array
- **cm** (*ClassManager*) – a ClassManager object

get_length()

get_obj()

get_raw()

get_size()

Return the number of elements in the array

Return type int

get_values()

Return a series of size encoded_value byte sequences in the format specified by this section, concatenated sequentially

Return type a list of *EncodedValue* objects

show()

class androguard.core.bytecodes.dvm.**EncodedArrayItem**(*buff*, *cm*)

Bases: `object`

This class can parse an encoded_array_item of a dex file

Parameters

- **buff** (*Buff object*) – a string which represents a Buff object of the encoded_array_item
- **cm** (*ClassManager*) – a ClassManager object

get_length()

get_obj()

get_off()

get_raw()

get_value()

Return the bytes representing the encoded array value

Return type a *EncodedArray* object

set_off(*off*)

show()

class androguard.core.bytecodes.dvm.**EncodedCatchHandler**(*buff*, *cm*)

Bases: object

This class can parse an encoded_catch_handler of a dex file

Parameters

- **buff** (*Buff object*) – a string which represents a Buff object of the encoded_catch_handler
- **cm** (*ClassManager*) – a ClassManager object

get_catch_all_addr()

Return the bytecode address of the catch-all handler. This element is only present if size is non-positive.

Return type int

get_handlers()

Return the stream of abs(size) encoded items, one for each caught type, in the order that the types should be tested.

Return type a list of *EncodedTypeAddrPair* objects

get_length()

get_off()

get_raw()

Return type bytearray

get_size()

Return the number of catch types in this list

Return type int

set_off(*off*)

show()

class androguard.core.bytecodes.dvm.**EncodedCatchHandlerList**(*buff*, *cm*)

Bases: object

This class can parse an encoded_catch_handler_list of a dex file

Parameters

- **buff** (*Buff object*) – a string which represents a Buff object of the encoded_catch_handler_list
- **cm** (*ClassManager*) – a ClassManager object

get_length()

get_list()

Return the actual list of handler lists, represented directly (not as offsets), and concatenated sequentially

Return type a list of *EncodedCatchHandler* objects

get_obj()

get_off()

get_raw()

Return type bytearray

get_size()
Return the size of this list, in entries

Return type int

set_off(*off*)

show()

class androguard.core.bytecodes.dvm.**EncodedField**(*buff*, *cm*)
Bases: object

This class can parse an encoded_field of a dex file

Parameters

- **buff** (*Buff object*) – a string which represents a Buff object of the encoded field
- **cm** (*ClassManager*) – a ClassManager object

adjust_idx(*val*)

get_access_flags()
Return the access flags of the field

Return type int

get_access_flags_string()
Return the access flags string of the field

Return type string

get_class_name()
Return the class name of the field

Return type string

get_descriptor()
Return the descriptor of the field

The descriptor of a field is the type of the field.

Return type string

get_field_idx()
Return the real index of the method

Return type int

get_field_idx_diff()
Return the index into the field_ids list for the identity of this field (includes the name and descriptor), represented as a difference from the index of previous element in the list

Return type int

get_init_value()
Return the init value object of the field

Return type *EncodedValue*

get_name()
Return the name of the field

Return type string

get_obj()

get_raw()

get_size()

load()

reload()

set_init_value(*value*)
Setup the init value object of the field

Parameters value (*EncodedValue*) – the init value

set_name(*value*)

show()
Display the information (with a pretty print) about the field

class androguard.core.bytecodes.dvm.**EncodedMethod**(*buff*, *cm*)
Bases: object

This class can parse an encoded_method of a dex file

Parameters

- **buff** (*Buff object*) – a string which represents a Buff object of the encoded_method
- **cm** (*ClassManager*) – a ClassManager object

access_flags = None
access flags of the method

add_inote(*msg*, *idx*, *off=None*)
Add a message to a specific instruction by using (default) the index of the address if specified

Parameters

- **msg** (*string*) – the message
- **idx** (*int*) – index of the instruction (the position in the list of the instruction)
- **off** (*int*) – address of the instruction

add_note(*msg*)
Add a message to this method

Parameters msg (*string*) – the message

adjust_idx(*val*)

code_off = None
offset of the code section

property descriptor
Get the descriptor of the method

each_params_by_register(*nb*, *proto*)
From the Dalvik Bytecode documentation:

> The N arguments to a method land in the last N registers > of the method's invocation frame, in order. > Wide arguments consume two registers. > Instance methods are passed a this reference as their first argument.

This method will print a description of the register usage to stdout.

Parameters

- **nb** – number of registers
- **proto** – descriptor of method

property full_name
Return class_name + name + descriptor, separated by spaces (no access flags

get_access_flags()
Return the access flags of the method

Return type int

get_access_flags_string()
Return the access flags string of the method

A description of all access flags can be found here: https://source.android.com/devices/tech/dalvik/dex-format#access-flags

Return type string

get_address()
Return the offset from the start of the file to the code structure for this method, or 0 if this method is either abstract or native

Return type int

get_class_name()
Return the class name of the method

Return type string

get_code()
Return the code object associated to the method

Return type *DalvikCode* object or None if no Code

get_code_off()
Return the offset from the start of the file to the code structure for this method, or 0 if this method is either abstract or native

Return type int

get_debug()
Return the debug object associated to this method

Return type *DebugInfoItem*

get_descriptor()
Return the descriptor of the method A method descriptor will have the form (A A A ...)R Where A are the arguments to the method and R is the return type. Basic types will have the short form, i.e. I for integer, V for void and class types will be named like a classname, e.g. Ljava/lang/String;.

Typical descriptors will look like this: ` (I)I // one integer argument, integer return (C)Z // one char argument, boolean as return (Ljava/lang/CharSequence; I)I // CharSequence and integer as argyument, integer as return (C)Ljava/lang/String; // char as argument, String as return. `

More information about type descriptors are found here: https://source.android.com/devices/tech/dalvik/dex-format#typedescriptor

Return type string

get_information()
Get brief information about the method's register use, parameters and return type.

The resulting dictionary has the form:

```
{
    registers: (start, end),
    params: [(reg_1, type_1), (reg_2, type_2), ..., (reg_n, type_n)],
    return: type
)
```

The end register is not the last register used, but the last register used not for parameters. Hence, they represent the local registers. The start register is always zero. The register numbers for the parameters can be found in the tuples for each parameter.

Returns a dictionary with the basic information about the method

Return type dict

`get_instruction``(`*idx*, *off=None*`)`

Get a particular instruction by using (default) the index of the address if specified

Parameters

- **`idx`** (*`int`*) – index of the instruction (the position in the list of the instruction)
- **`off`** (*`int`*) – address of the instruction

Return type an *`Instruction`* object

`get_instructions``()`

Get the instructions

Return type a generator of each *`Instruction`* (or a cached list of instructions if you have setup instructions)

`get_instructions_idx``()`

Iterate over all instructions of the method, but also return the current index. This is the same as using *`get_instructions()`* and adding the instruction length to a variable each time.

Returns

Return type Iterator[(int, *Instruction*)]

`get_length``()`

Return the length of the associated code of the method

Return type int

`get_locals``()`

Get the number of local registers used by the method

This number is equal to the number of registers minus the number of parameters minus 1.

Returns number of local registers

Return type int

`get_method_idx``()`

Return the real index of the method

Return type int

`get_method_idx_diff``()`

Return index into the method_ids list for the identity of this method (includes the name and descriptor), represented as a difference from the index of previous element in the lis

Return type int

get_name()

Return the name of the method

Return type string

get_raw()

get_short_string()

Return a shorter formatted String which encodes this method. The returned name has the form: <classname> <methodname> ([arguments …])<returntype>

- All Class names are condensed to the actual name (no package).
- Access flags are not returned.
- <init> and <clinit> are NOT replaced by the classname!

This name might not be unique!

Returns str

get_size()

get_source()

get_triple()

is_cached_instructions()

load()

method_idx_diff = None

method index diff in the corresponding section

reload()

set_code_idx(*idx*)

Set the start address of the buffer to disassemble

Parameters **idx** (*int*) – the index

set_instructions(*instructions*)

Set the instructions

Parameters **instructions** (a list of *Instruction*) – the list of instructions

set_name(*value*)

show()

Display the information (with a pretty print) about the method

show_info()

Display the basic information about the method

show_notes()

Display the notes about the method

source()

Return the source code of this method

Return type string

class androguard.core.bytecodes.dvm.**EncodedTypeAddrPair**(*cm*, *buff*)

Bases: object

This class can parse an encoded_type_addr_pair of a dex file

Parameters

- **buff** (*Buff object*) – a string which represents a Buff object of the encoded_type_addr_pair
- **cm** (*ClassManager*) – a ClassManager object

get_addr()

Return the bytecode address of the associated exception handler

Return type int

get_length()

get_obj()

get_raw()

get_type_idx()

Return the index into the type_ids list for the type of the exception to catch

Return type int

show()

class androguard.core.bytecodes.dvm.**EncodedValue**(*buff*, *cm*)

Bases: object

This class can parse an encoded_value of a dex file

Parameters

- **buff** (*Buff object*) – a string which represents a Buff object of the encoded_value
- **cm** (*ClassManager*) – a ClassManager object

get_length()

get_obj()

get_raw()

get_value()

Return the bytes representing the value, variable in length and interpreted differently for different value_type bytes, though always little-endian

Return type an object representing the value

get_value_arg()

get_value_type()

show()

class androguard.core.bytecodes.dvm.**ExportObject**

Bases: object

Wrapper object for ipython exports

class androguard.core.bytecodes.dvm.**FieldAnnotation**(*buff*, *cm*)

Bases: object

This class can parse a field_annotation of a dex file

Parameters

- **buff** (*Buff object*) – a string which represents a Buff object of the field_annotation
- **cm** (*ClassManager*) – a ClassManager object

get_annotations_off()

Return the offset from the start of the file to the list of annotations for the field

Return type int

get_field_idx()

Return the index into the field_ids list for the identity of the field being annotated

Return type int

get_length()

get_obj()

get_off()

get_raw()

set_off(*off*)

show()

class androguard.core.bytecodes.dvm.**FieldHIdItem**(*size*, *buff*, *cm*)

Bases: object

This class can parse a list of field_id_item of a dex file

Parameters

- **buff** (*Buff object*) – a string which represents a Buff object of the list of field_id_item
- **cm** (*ClassManager*) – a ClassManager object

get(*idx*)

get_length()

get_obj()

get_off()

get_raw()

gets()

set_off(*off*)

show()

class androguard.core.bytecodes.dvm.**FieldIdItem**(*buff*, *cm*)

Bases: object

This class can parse a field_id_item of a dex file

Parameters

- **buff** (*Buff object*) – a string which represents a Buff object of the field_id_item
- **cm** (*ClassManager*) – a ClassManager object

get_class_idx()

Return the index into the type_ids list for the definer of this field

Return type int

get_class_name()

Return the class name of the field

Return type string

get_descriptor()

Return the descriptor of the field

Return type string

get_length()

get_list()

get_name()

Return the name of the field

Return type string

get_name_idx()

Return the index into the string_ids list for the name of this field

Return type int

get_obj()

get_raw()

get_type()

Return the type of the field

Return type string

get_type_idx()

Return the index into the type_ids list for the type of this field

Return type int

reload()

show()

class androguard.core.bytecodes.dvm.**FieldIdItemInvalid**

Bases: `object`

get_class_name()

get_descriptor()

get_list()

get_name()

get_type()

show()

class androguard.core.bytecodes.dvm.**FillArrayData**(*cm*, *buff*)

Bases: `object`

This class can parse a FillArrayData instruction

Parameters **buff** – a Buff object which represents a buffer where the instruction is stored

add_note(*msg*)

Add a note to this instruction

Parameters **msg** (*objects (string)*) – the message

disasm()

get_data()

Return the data of this instruction (the payload)

Return type bytes

get_formatted_operands()

get_hex()
Returns a HEX String, separated by spaces every byte

get_length()
Return the length of the instruction

Return type int

get_name()
Return the name of the instruction

Return type string

get_notes()
Get all notes from this instruction

Return type a list of objects

get_op_value()
Get the value of the opcode

Return type int

get_operands(*idx=-1*)

get_output(*idx=-1*)
Return an additional output of the instruction

Return type string

get_raw()

show(*pos*)
Print the instruction

show_buff(*pos*)
Return the display of the instruction

Return type string

class androguard.core.bytecodes.dvm.**HeaderItem**(*size*, *buff*, *cm*)
Bases: object

This class can parse an header_item of a dex file. Several checks are performed to detect if this is not an header_item. Also the Adler32 checksum of the file is calculated in order to detect file corruption. :param buff: a string which represents a Buff object of the header_item :type androguard.core.bytecode.BuffHandle buff: Buff object :param cm: a ClassManager object :type cm: ClassManager

get_length()

get_obj()

get_off()

get_raw()

set_off(*off*)

show()

class androguard.core.bytecodes.dvm.**Instruction**
Bases: object

This class represents a Dalvik instruction

It can both handle normal instructions as well as optimized instructions.

> **Warning:** There is not much documentation about the optimized opcodes! Hence, it relies on reverese engineered specification!

More information about the instruction format can be found in the official documentation: https://source.android.com/devices/tech/dalvik/instruction-formats.html

> **Warning:** Values stored in the instructions are already interpreted at this stage.

The Dalvik VM has a eight opcodes to create constant integer values. There are four variants for 32bit values and four for 64bit. If floating point numbers are required, you have to use the conversion opcodes like `int-to-float`, `int-to-double` or the variants using `long`.

Androguard will always show the values as they are used in the opcode and also extend signs and shift values! As an example: The opcode `const/high16` can be used to create constant values where the lower 16 bits are all zero. In this case, androguard will process bytecode `15 00 CD AB` as beeing `const/high16 v0, 0xABCD0000`. For the sign-extension, nothing is really done here, as it only affects the bit represenation in the virtual machine. As androguard parses the values and uses python types internally, we are not bound to specific size.

OP = 0

disasm()

Some small line for disassembly view

get_formatted_operands()

Returns the formatted operands, if any. This is a list with the parsed and interpreted operands of the opcode.

Returns None if no operands, otherwise a List

Deprecated since version 3.4.0: Will be removed! This method always returns None

get_hex()

Returns a HEX String, separated by spaces every byte

The hex string contains the raw bytes of the instruction, including the opcode and all arguments.

Return type str

get_kind()

Return the 'kind' argument of the instruction

This is the type of the argument, i.e. in which kind of table you have to look up the argument in the ClassManager

Return type int

get_length()

Return the length of the instruction in bytes

Return type int

get_literals()

Return the associated literals

Return type list of int

get_name()
Return the mnemonic of the instruction

Return type string

get_op_value()
Return the numerical value of the opcode

Return type int

get_operands(*idx=-1*)
Return all operands

This will return a list of tuples, containing the Enum `Operand` at the first position and the objects afterwards.

Return type List[Tuple(Operand, object, ..)]

get_output(*idx=-1*)
Return an additional output of the instruction

Return type string

get_raw()
Return the object in a raw format

Return type string

get_ref_kind()
Return the value of the 'kind' argument

Return type value

get_translated_kind()
Return the translated value of the 'kind' argument

Return type string

length = 0

show(*idx*)
Print the instruction

No Line ending is printed.

show_buff(*idx*)
Return the display of the instruction

Return type string

class androguard.core.bytecodes.dvm.**Instruction00x**(*cm*, *buff*)
Bases: *androguard.core.bytecodes.dvm.Instruction*

A class for unused instructions, has zero length and raises an error on initialization

length = 0

class androguard.core.bytecodes.dvm.**Instruction10t**(*cm*, *buff*)
Bases: *androguard.core.bytecodes.dvm.Instruction*

This class represents all instructions which have the 10t format

get_operands(*idx=-1*)
Return all operands

This will return a list of tuples, containing the Enum `Operand` at the first position and the objects afterwards.

Return type List[Tuple(Operand, object, ..)]

get_output(*idx=-1*)
Return an additional output of the instruction

Return type string

get_raw()
Return the object in a raw format

Return type string

get_ref_off()

length = 2

class androguard.core.bytecodes.dvm.**Instruction10x**(*cm*, *buff*)
Bases: *androguard.core.bytecodes.dvm.Instruction*

This class represents all instructions which have the 10x format

get_raw()
Return the object in a raw format

Return type string

length = 2

class androguard.core.bytecodes.dvm.**Instruction11n**(*cm*, *buff*)
Bases: *androguard.core.bytecodes.dvm.Instruction*

This class represents all instructions which have the 11n format

get_literals()
Return the associated literals

Return type list of int

get_operands(*idx=-1*)
Return all operands

This will return a list of tuples, containing the Enum `Operand` at the first position and the objects afterwards.

Return type List[Tuple(Operand, object, ..)]

get_output(*idx=-1*)
Return an additional output of the instruction

Return type string

get_raw()
Return the object in a raw format

Return type string

length = 2

class androguard.core.bytecodes.dvm.**Instruction11x**(*cm*, *buff*)
Bases: *androguard.core.bytecodes.dvm.Instruction*

This class represents all instructions which have the 11x format

get_operands(*idx=-1*)
Return all operands

This will return a list of tuples, containing the Enum `Operand` at the first position and the objects afterwards.

Return type List[Tuple(Operand, object, ..)]

get_output(*idx=-1*)
Return an additional output of the instruction

Return type string

get_raw()
Return the object in a raw format

Return type string

length = 2

class androguard.core.bytecodes.dvm.**Instruction12x**(*cm*, *buff*)
Bases: *androguard.core.bytecodes.dvm.Instruction*

This class represents all instructions which have the 12x format

get_operands(*idx=-1*)
Return all operands

This will return a list of tuples, containing the Enum `Operand` at the first position and the objects afterwards.

Return type List[Tuple(Operand, object, ..)]

get_output(*idx=-1*)
Return an additional output of the instruction

Return type string

get_raw()
Return the object in a raw format

Return type string

length = 2

class androguard.core.bytecodes.dvm.**Instruction20bc**(*cm*, *buff*)
Bases: *androguard.core.bytecodes.dvm.Instruction*

This class represents all instructions which have the 20bc format

get_operands(*idx=-1*)
Return all operands

This will return a list of tuples, containing the Enum `Operand` at the first position and the objects afterwards.

Return type List[Tuple(Operand, object, ..)]

get_output(*idx=-1*)
Return an additional output of the instruction

Return type string

get_raw()
Return the object in a raw format

Return type string

length = 4

class androguard.core.bytecodes.dvm.**Instruction20t**(*cm*, *buff*)

Bases: *androguard.core.bytecodes.dvm.Instruction*

This class represents all instructions which have the 20t format

get_operands(*idx=-1*)

Return all operands

This will return a list of tuples, containing the Enum `Operand` at the first position and the objects afterwards.

Return type List[Tuple(Operand, object, ..)]

get_output(*idx=-1*)

Return an additional output of the instruction

Return type string

get_raw()

Return the object in a raw format

Return type string

get_ref_off()

length = 4

class androguard.core.bytecodes.dvm.**Instruction21c**(*cm*, *buff*)

Bases: *androguard.core.bytecodes.dvm.Instruction*

This class represents all instructions which have the 21c format

get_operands(*idx=-1*)

Return all operands

This will return a list of tuples, containing the Enum `Operand` at the first position and the objects afterwards.

Return type List[Tuple(Operand, object, ..)]

get_output(*idx=-1*)

Return an additional output of the instruction

Return type string

get_raw()

Return the object in a raw format

Return type string

get_raw_string()

get_ref_kind()

Return the value of the 'kind' argument

Return type value

get_string()

length = 4

class androguard.core.bytecodes.dvm.**Instruction21h**(*cm*, *buff*)

Bases: *androguard.core.bytecodes.dvm.Instruction*

This class represents all instructions which have the 21h format

get_literals()
Return the associated literals

Return type list of int

get_operands(*idx=-1*)
Return all operands

This will return a list of tuples, containing the Enum `Operand` at the first position and the objects afterwards.

Return type List[Tuple(Operand, object, ..)]

get_output(*idx=-1*)
Return an additional output of the instruction

Return type string

get_raw()
Return the object in a raw format

Return type string

length = 4

class androguard.core.bytecodes.dvm.**Instruction21s**(*cm*, *buff*)
Bases: *androguard.core.bytecodes.dvm.Instruction*

This class represents all instructions which have the 21s format

get_literals()
Return the associated literals

Return type list of int

get_operands(*idx=-1*)
Return all operands

This will return a list of tuples, containing the Enum `Operand` at the first position and the objects afterwards.

Return type List[Tuple(Operand, object, ..)]

get_output(*idx=-1*)
Return an additional output of the instruction

Return type string

get_raw()
Return the object in a raw format

Return type string

length = 4

class androguard.core.bytecodes.dvm.**Instruction21t**(*cm*, *buff*)
Bases: *androguard.core.bytecodes.dvm.Instruction*

This class represents all instructions which have the 21t format

get_operands(*idx=-1*)
Return all operands

This will return a list of tuples, containing the Enum `Operand` at the first position and the objects afterwards.

Return type List[Tuple(Operand, object, ..)]

get_output(*idx=-1*)
Return an additional output of the instruction

Return type string

get_raw()
Return the object in a raw format

Return type string

get_ref_off()

length = 4

class androguard.core.bytecodes.dvm.**Instruction22b**(*cm*, *buff*)
Bases: *androguard.core.bytecodes.dvm.Instruction*

This class represents all instructions which have the 22b format

get_literals()
Return the associated literals

Return type list of int

get_operands(*idx=-1*)
Return all operands

This will return a list of tuples, containing the Enum `Operand` at the first position and the objects afterwards.

Return type List[Tuple(Operand, object, ..)]

get_output(*idx=-1*)
Return an additional output of the instruction

Return type string

get_raw()
Return the object in a raw format

Return type string

length = 4

class androguard.core.bytecodes.dvm.**Instruction22c**(*cm*, *buff*)
Bases: *androguard.core.bytecodes.dvm.Instruction*

This class represents all instructions which have the 22c format

get_operands(*idx=-1*)
Return all operands

This will return a list of tuples, containing the Enum `Operand` at the first position and the objects afterwards.

Return type List[Tuple(Operand, object, ..)]

get_output(*idx=-1*)
Return an additional output of the instruction

Return type string

get_raw()
Return the object in a raw format

Return type string

get_ref_kind()
Return the value of the 'kind' argument

Return type value

length = 4

class androguard.core.bytecodes.dvm.**Instruction22cs**(*cm*, *buff*)
Bases: *androguard.core.bytecodes.dvm.Instruction*

This class represents all instructions which have the 22cs format

get_operands(*idx=-1*)
Return all operands

This will return a list of tuples, containing the Enum `Operand` at the first position and the objects afterwards.

Return type List[Tuple(Operand, object, ..)]

get_output(*idx=-1*)
Return an additional output of the instruction

Return type string

get_raw()
Return the object in a raw format

Return type string

get_ref_kind()
Return the value of the 'kind' argument

Return type value

length = 4

class androguard.core.bytecodes.dvm.**Instruction22s**(*cm*, *buff*)
Bases: *androguard.core.bytecodes.dvm.Instruction*

This class represents all instructions which have the 22s format

get_literals()
Return the associated literals

Return type list of int

get_operands(*idx=-1*)
Return all operands

This will return a list of tuples, containing the Enum `Operand` at the first position and the objects afterwards.

Return type List[Tuple(Operand, object, ..)]

get_output(*idx=-1*)
Return an additional output of the instruction

Return type string

get_raw()
Return the object in a raw format

Return type string

length = 4

class androguard.core.bytecodes.dvm.**Instruction22t**(*cm*, *buff*)

Bases: *androguard.core.bytecodes.dvm.Instruction*

This class represents all instructions which have the 22t format

get_operands(*idx=-1*)

Return all operands

This will return a list of tuples, containing the Enum `Operand` at the first position and the objects afterwards.

Return type List[Tuple(Operand, object, ..)]

get_output(*idx=-1*)

Return an additional output of the instruction

Return type string

get_raw()

Return the object in a raw format

Return type string

get_ref_off()

length = 4

class androguard.core.bytecodes.dvm.**Instruction22x**(*cm*, *buff*)

Bases: *androguard.core.bytecodes.dvm.Instruction*

This class represents all instructions which have the 22x format

get_operands(*idx=-1*)

Return all operands

This will return a list of tuples, containing the Enum `Operand` at the first position and the objects afterwards.

Return type List[Tuple(Operand, object, ..)]

get_output(*idx=-1*)

Return an additional output of the instruction

Return type string

get_raw()

Return the object in a raw format

Return type string

length = 4

class androguard.core.bytecodes.dvm.**Instruction23x**(*cm*, *buff*)

Bases: *androguard.core.bytecodes.dvm.Instruction*

This class represents all instructions which have the 23x format

get_operands(*idx=-1*)

Return all operands

This will return a list of tuples, containing the Enum `Operand` at the first position and the objects afterwards.

Return type List[Tuple(Operand, object, ..)]

get_output(*idx=-1*)

Return an additional output of the instruction

Return type string

get_raw()
Return the object in a raw format

Return type string

length = 4

class androguard.core.bytecodes.dvm.**Instruction30t**(*cm*, *buff*)
Bases: *androguard.core.bytecodes.dvm.Instruction*

This class represents all instructions which have the 30t format

get_operands(*idx=-1*)
Return all operands

This will return a list of tuples, containing the Enum `Operand` at the first position and the objects afterwards.

Return type List[Tuple(Operand, object, ..)]

get_output(*idx=-1*)
Return an additional output of the instruction

Return type string

get_raw()
Return the object in a raw format

Return type string

get_ref_off()

length = 6

class androguard.core.bytecodes.dvm.**Instruction31c**(*cm*, *buff*)
Bases: *androguard.core.bytecodes.dvm.Instruction*

This class represents all instructions which have the 31c format

get_operands(*idx=-1*)
Return all operands

This will return a list of tuples, containing the Enum `Operand` at the first position and the objects afterwards.

Return type List[Tuple(Operand, object, ..)]

get_output(*idx=-1*)
Return an additional output of the instruction

Return type string

get_raw()
Return the object in a raw format

Return type string

get_raw_string()

get_ref_kind()
Return the value of the 'kind' argument

Return type value

get_string()
Return the string associated to the 'kind' argument

Return type string

length = 6

class androguard.core.bytecodes.dvm.**Instruction31i**(*cm*, *buff*)
Bases: *androguard.core.bytecodes.dvm.Instruction*

This class represents all instructions which have the 31i format

get_literals()
Return the associated literals

Return type list of int

get_operands(*idx=-1*)
Return all operands

This will return a list of tuples, containing the Enum `Operand` at the first position and the objects afterwards.

Return type List[Tuple(Operand, object, ..)]

get_output(*idx=-1*)
Return an additional output of the instruction

Return type string

get_raw()
Return the object in a raw format

Return type string

length = 6

class androguard.core.bytecodes.dvm.**Instruction31t**(*cm*, *buff*)
Bases: *androguard.core.bytecodes.dvm.Instruction*

This class represents all instructions which have the 31t format

get_operands(*idx=-1*)
Return all operands

This will return a list of tuples, containing the Enum `Operand` at the first position and the objects afterwards.

Return type List[Tuple(Operand, object, ..)]

get_output(*idx=-1*)
Return an additional output of the instruction

Return type string

get_raw()
Return the object in a raw format

Return type string

get_ref_off()

length = 6

class androguard.core.bytecodes.dvm.**Instruction32x**(*cm*, *buff*)
Bases: *androguard.core.bytecodes.dvm.Instruction*

This class represents all instructions which have the 32x format

get_operands(*idx=-1*)
Return all operands

This will return a list of tuples, containing the Enum `Operand` at the first position and the objects afterwards.

Return type List[Tuple(Operand, object, ..)]

get_output(*idx=-1*)
Return an additional output of the instruction

Return type string

get_raw()
Return the object in a raw format

Return type string

length = 6

class androguard.core.bytecodes.dvm.**Instruction35c**(*cm*, *buff*)
Bases: *androguard.core.bytecodes.dvm.Instruction*

This class represents all instructions which have the 35c format

get_operands(*idx=-1*)
Return all operands

This will return a list of tuples, containing the Enum `Operand` at the first position and the objects afterwards.

Return type List[Tuple(Operand, object, ..)]

get_output(*idx=-1*)
Return an additional output of the instruction

Return type string

get_raw()
Return the object in a raw format

Return type string

get_ref_kind()
Return the value of the 'kind' argument

Return type value

length = 6

class androguard.core.bytecodes.dvm.**Instruction35mi**(*cm*, *buff*)
Bases: *androguard.core.bytecodes.dvm.Instruction*

This class represents all instructions which have the 35mi format

get_operands(*idx=-1*)
Return all operands

This will return a list of tuples, containing the Enum `Operand` at the first position and the objects afterwards.

Return type List[Tuple(Operand, object, ..)]

get_output(*idx=-1*)
Return an additional output of the instruction

Return type string

get_raw()
Return the object in a raw format

Return type string

get_ref_kind()
Return the value of the 'kind' argument

Return type value

length = 6

class androguard.core.bytecodes.dvm.**Instruction35ms**(*cm*, *buff*)
Bases: *androguard.core.bytecodes.dvm.Instruction*

This class represents all instructions which have the 35ms format

get_operands(*idx=-1*)
Return all operands

This will return a list of tuples, containing the Enum `Operand` at the first position and the objects afterwards.

Return type List[Tuple(Operand, object, ..)]

get_output(*idx=-1*)
Return an additional output of the instruction

Return type string

get_raw()
Return the object in a raw format

Return type string

get_ref_kind()
Return the value of the 'kind' argument

Return type value

length = 6

class androguard.core.bytecodes.dvm.**Instruction3rc**(*cm*, *buff*)
Bases: *androguard.core.bytecodes.dvm.Instruction*

This class represents all instructions which have the 3rc format

get_operands(*idx=-1*)
Return all operands

This will return a list of tuples, containing the Enum `Operand` at the first position and the objects afterwards.

Return type List[Tuple(Operand, object, ..)]

get_output(*idx=-1*)
Return an additional output of the instruction

Return type string

get_raw()
Return the object in a raw format

Return type string

get_ref_kind()

Return the value of the 'kind' argument

Return type value

length = 6

class androguard.core.bytecodes.dvm.**Instruction3rmi**(*cm*, *buff*)

Bases: *androguard.core.bytecodes.dvm.Instruction*

This class represents all instructions which have the 3rmi format

Note, this instruction is similar to 3rc but holds an inline

get_operands(*idx=-1*)

Return all operands

This will return a list of tuples, containing the Enum `Operand` at the first position and the objects afterwards.

Return type List[Tuple(Operand, object, ..)]

get_output(*idx=-1*)

Return an additional output of the instruction

Return type string

get_raw()

Return the object in a raw format

Return type string

get_ref_kind()

Return the value of the 'kind' argument

Return type value

length = 6

class androguard.core.bytecodes.dvm.**Instruction3rms**(*cm*, *buff*)

Bases: *androguard.core.bytecodes.dvm.Instruction*

This class represents all instructions which have the 3rms format

Note, this instruction is similar to 3rc but holds a vtaboff

get_operands(*idx=-1*)

Return all operands

This will return a list of tuples, containing the Enum `Operand` at the first position and the objects afterwards.

Return type List[Tuple(Operand, object, ..)]

get_output(*idx=-1*)

Return an additional output of the instruction

Return type string

get_raw()

Return the object in a raw format

Return type string

get_ref_kind()

Return the value of the 'kind' argument

Return type value

length = 6

class androguard.core.bytecodes.dvm.**Instruction40sc**(*cm*, *buff*)

Bases: *androguard.core.bytecodes.dvm.Instruction*

This class represents all instructions which have the 40sc format

This instruction is only used in ODEX

get_operands(*idx=-1*)

Return all operands

This will return a list of tuples, containing the Enum `Operand` at the first position and the objects afterwards.

Return type List[Tuple(Operand, object, ..)]

get_output(*idx=-1*)

Return an additional output of the instruction

Return type string

get_raw()

Return the object in a raw format

Return type string

get_ref_kind()

Return the value of the 'kind' argument

Return type value

length = 8

class androguard.core.bytecodes.dvm.**Instruction41c**(*cm*, *buff*)

Bases: *androguard.core.bytecodes.dvm.Instruction*

This class represents all instructions which have the 41c format

This instruction is only used in ODEX

get_operands(*idx=-1*)

Return all operands

This will return a list of tuples, containing the Enum `Operand` at the first position and the objects afterwards.

Return type List[Tuple(Operand, object, ..)]

get_output(*idx=-1*)

Return an additional output of the instruction

Return type string

get_raw()

Return the object in a raw format

Return type string

get_ref_kind()

Return the value of the 'kind' argument

Return type value

length = 8

class androguard.core.bytecodes.dvm.**Instruction45cc**(*cm*, *buff*)

Bases: *androguard.core.bytecodes.dvm.Instruction*

get_operands()

Return all operands

This will return a list of tuples, containing the Enum `Operand` at the first position and the objects afterwards.

Return type List[Tuple(Operand, object, ..)]

get_output(*idx=-1*)

Return an additional output of the instruction

Return type string

get_raw()

Return the object in a raw format

Return type string

length = 8

class androguard.core.bytecodes.dvm.**Instruction4rcc**(*cm*, *buff*)

Bases: *androguard.core.bytecodes.dvm.Instruction*

get_operands()

Return all operands

This will return a list of tuples, containing the Enum `Operand` at the first position and the objects afterwards.

Return type List[Tuple(Operand, object, ..)]

get_output(*idx=-1*)

Return an additional output of the instruction

Return type string

get_raw()

Return the object in a raw format

Return type string

length = 8

class androguard.core.bytecodes.dvm.**Instruction51l**(*cm*, *buff*)

Bases: *androguard.core.bytecodes.dvm.Instruction*

This class represents all instructions which have the 51l format

get_literals()

Return the associated literals

Return type list of int

get_operands(*idx=-1*)

Return all operands

This will return a list of tuples, containing the Enum `Operand` at the first position and the objects afterwards.

Return type List[Tuple(Operand, object, ..)]

get_output(*idx=-1*)

Return an additional output of the instruction

Return type string

get_raw()
Return the object in a raw format

Return type string

length = 10

class androguard.core.bytecodes.dvm.**Instruction52c**(*cm*, *buff*)
Bases: *androguard.core.bytecodes.dvm.Instruction*

This class represents all instructions which have the 52c format

This instruction is only used in ODEX

get_operands(*idx=-1*)
Return all operands

This will return a list of tuples, containing the Enum `Operand` at the first position and the objects afterwards.

Return type List[Tuple(Operand, object, ..)]

get_output(*idx=-1*)
Return an additional output of the instruction

Return type string

get_raw()
Return the object in a raw format

Return type string

get_ref_kind()
Return the value of the 'kind' argument

Return type value

length = 10

class androguard.core.bytecodes.dvm.**Instruction5rc**(*cm*, *buff*)
Bases: *androguard.core.bytecodes.dvm.Instruction*

This class represents all instructions which have the 5rc format

This instruction is only used in ODEX

get_operands(*idx=-1*)
Return all operands

This will return a list of tuples, containing the Enum `Operand` at the first position and the objects afterwards.

Return type List[Tuple(Operand, object, ..)]

get_output(*idx=-1*)
Return an additional output of the instruction

Return type string

get_raw()
Return the object in a raw format

Return type string

get_ref_kind()

Return the value of the 'kind' argument

Return type value

length = 10

exception androguard.core.bytecodes.dvm.**InvalidInstruction**

Bases: Exception

class androguard.core.bytecodes.dvm.**LinearSweepAlgorithm**

Bases: object

This class is used to disassemble a method. The algorithm used by this class is linear sweep.

static get_instructions(*cm*, *size*, *insn*, *idx*)

Yields all instructions for the given bytecode sequence. If unknown/corrupt/unused instructions are encountered, the loop will stop and an error is written to the log.

That means that the bytecode read might be corrupt or was crafted in this way, to break parsers.

Parameters

- **cm** (ClassManager) – a ClassManager object
- **size** (*int*) – the total size of the buffer in 16-bit units
- **insn** (*bytearray*) – a raw buffer where are the instructions
- **idx** (*int*) – a start address in the buffer
- **raise_errors** (*bool*) – True to raise errors instead of simply logging them

Return type Iterator[*Instruction*]

class androguard.core.bytecodes.dvm.**MapItem**(*buff*, *cm*)

Bases: object

get_item()

Return the associated item itself. Might return None, if *parse()* was not called yet.

This method is the same as *get_item()*.

get_length()

get_obj()

Return the associated item itself. Might return None, if *parse()* was not called yet.

This method is the same as *get_item()*.

get_off()

Gets the offset of the map item itself inside the DEX file

get_offset()

Gets the offset of the item of the map item

get_raw()

get_size()

Returns the number of items found at the location indicated by *get_offset()*.

get_type()

parse()

set_item(*item*)

show()

class androguard.core.bytecodes.dvm.**MapList**(*cm*, *off*, *buff*)

Bases: object

This class can parse the "map_list" of the dex format

https://source.android.com/devices/tech/dalvik/dex-format#map-list

get_class_manager()

get_item_type(*ttype*)

Get a particular item type

Parameters **ttype** – a string which represents the desired type

Return type None or the item object

get_length()

get_obj()

get_off()

get_raw()

set_off(*off*)

show()

Print with a pretty display the MapList object

class androguard.core.bytecodes.dvm.**MethodAnnotation**(*buff*, *cm*)

Bases: object

This class can parse a method_annotation of a dex file

Parameters

- **buff** (*Buff object*) – a string which represents a Buff object of the method_annotation
- **cm** (*ClassManager*) – a ClassManager object

get_annotations_off()

Return the offset from the start of the file to the list of annotations for the method

Return type int

get_length()

get_method_idx()

Return the index into the method_ids list for the identity of the method being annotated

Return type int

get_obj()

get_off()

get_raw()

set_off(*off*)

show()

class androguard.core.bytecodes.dvm.**MethodHIdItem**(*size*, *buff*, *cm*)

Bases: object

This class can parse a list of method_id_item of a dex file

Parameters

- **buff** (*Buff object*) – a string which represents a Buff object of the list of method_id_item
- **cm** (*ClassManager*) – a ClassManager object

get(*idx*)

get_length()

get_obj()

get_off()

get_raw()

reload()

set_off(*off*)

show()

class androguard.core.bytecodes.dvm.**MethodIdItem**(*buff*, *cm*)

Bases: object

This class can parse a method_id_item of a dex file

Parameters

- **buff** (*Buff object*) – a string which represents a Buff object of the method_id_item
- **cm** (*ClassManager*) – a ClassManager object

get_class_idx()

Return the index into the type_ids list for the definer of this method

Return type int

get_class_name()

Return the class name of the method

Return type string

get_descriptor()

Return the descriptor

Return type string

get_length()

get_list()

get_name()

Return the name of the method

Return type string

get_name_idx()

Return the index into the string_ids list for the name of this method

Return type int

get_obj()

get_proto()

Return the prototype of the method

Return type string

get_proto_idx()
Return the index into the proto_ids list for the prototype of this method

Return type int

get_raw()

get_real_descriptor()
Return the real descriptor (i.e. without extra spaces)

Return type string

get_triple()

reload()

show()

class androguard.core.bytecodes.dvm.**MethodIdItemInvalid**
Bases: object

get_class_name()

get_descriptor()

get_list()

get_name()

get_proto()

show()

class androguard.core.bytecodes.dvm.**OdexDependencies**(*buff*)
Bases: object

This class can parse the odex dependencies

Parameters **buff** – a Buff object string which represents the odex dependencies

get_dependencies()
Return the list of dependencies

Return type a list of strings

get_raw()

class androguard.core.bytecodes.dvm.**OdexHeaderItem**(*buff*)
Bases: object

This class can parse the odex header

Parameters **buff** – a Buff object string which represents the odex dependencies

get_raw()

show()

class androguard.core.bytecodes.dvm.**OffObj**(*o*)
Bases: object

class androguard.core.bytecodes.dvm.**PackedSwitch**(*cm*, *buff*)
Bases: object

This class can parse a PackedSwitch instruction

Parameters **buff** – a Buff object which represents a buffer where the instruction is stored

add_note(*msg*)
Add a note to this instruction

Parameters **msg** (*objects (string)*) – the message

disasm()

get_formatted_operands()

get_hex()
Returns a HEX String, separated by spaces every byte

get_keys()
Return the keys of the instruction

Return type a list of long

get_length()

get_name()
Return the name of the instruction

Return type string

get_notes()
Get all notes from this instruction

Return type a list of objects

get_op_value()
Get the value of the opcode

Return type int

get_operands(*idx=-1*)
Return an additional output of the instruction

Return type string

get_output(*idx=-1*)
Return an additional output of the instruction

rtype string

get_raw()

get_targets()
Return the targets (address) of the instruction

Return type a list of long

get_values()

show(*pos*)
Print the instruction

show_buff(*pos*)
Return the display of the instruction

Return type string

class androguard.core.bytecodes.dvm.**ParameterAnnotation**(*buff*, *cm*)
Bases: object

This class can parse a parameter_annotation of a dex file

Parameters

- **buff** (*Buff object*) – a string which represents a Buff object of the parameter_annotation
- **cm** (*ClassManager*) – a ClassManager object

get_annotations_off()

Return the offset from the start of the file to the list of annotations for the method parameters

Return type int

get_length()

get_method_idx()

Return the index into the method_ids list for the identity of the method whose parameters are being annotated

Return type int

get_obj()

get_off()

get_raw()

set_off(*off*)

show()

class androguard.core.bytecodes.dvm.**ProtoHIdItem**(*size*, *buff*, *cm*)

Bases: object

This class can parse a list of proto_id_item of a dex file

Parameters

- **buff** (*Buff object*) – a string which represents a Buff object of the list of proto_id_item
- **cm** (*ClassManager*) – a ClassManager object

get(*idx*)

get_length()

get_obj()

get_off()

get_raw()

set_off(*off*)

show()

class androguard.core.bytecodes.dvm.**ProtoIdItem**(*buff*, *cm*)

Bases: object

This class can parse a proto_id_item of a dex file

Parameters

- **buff** (*Buff object*) – a string which represents a Buff object of the proto_id_item
- **cm** (*ClassManager*) – a ClassManager object

get_length()

get_obj()

get_parameters_off()

Return the offset from the start of the file to the list of parameter types for this prototype, or 0 if this prototype has no parameters

Return type int

get_parameters_off_value()

Return the string associated to the parameters_off

Return type *MUTF8String*

get_raw()

get_return_type_idx()

Return the index into the type_ids list for the return type of this prototype

Return type int

get_return_type_idx_value()

Return the string associated to the return_type_idx

Return type string

get_shorty_idx()

Return the index into the string_ids list for the short-form descriptor string of this prototype

Return type int

get_shorty_idx_value()

Return the string associated to the shorty_idx

Return type string

show()

class androguard.core.bytecodes.dvm.**ProtoIdItemInvalid**

Bases: `object`

get_params()

get_return_type()

get_shorty()

show()

class androguard.core.bytecodes.dvm.**SparseSwitch**(*cm*, *buff*)

Bases: `object`

This class can parse a SparseSwitch instruction

Parameters **buff** – a Buff object which represents a buffer where the instruction is stored

add_note(*msg*)

Add a note to this instruction

Parameters **msg** (*objects (string)*) – the message

disasm()

get_formatted_operands()

get_hex()

Returns a HEX String, separated by spaces every byte

get_keys()

Return the keys of the instruction

Return type a list of long

get_length()

get_name()
Return the name of the instruction

Return type string

get_notes()
Get all notes from this instruction

Return type a list of objects

get_op_value()
Get the value of the opcode

Return type int

get_operands(*idx=-1*)
Return an additional output of the instruction

Return type string

get_output(*idx=-1*)
Return an additional output of the instruction

Return type string

get_raw()

get_targets()
Return the targets (address) of the instruction

Return type a list of long

get_values()

show(*pos*)
Print the instruction

show_buff(*pos*)
Return the display of the instruction

Return type string

class androguard.core.bytecodes.dvm.**StringDataItem**(*buff*, *cm*)
Bases: object

This class can parse a string_data_item of a dex file

Strings in Dalvik files might not be representable in python! This is due to the fact, that you can store any UTF-16 character inside a Dalvik file, but this string might not be decodeable in python as it can contain invalid surrogate-pairs.

To circumvent this issue, this class has different methods how to access the string. There are also some fallbacks implemented to make a "invalid" string printable in python. Dalvik uses MUTF-8 as encoding for the strings. This encoding has the advantage to allow for null terminated strings in UTF-8 encoding, as the null character maps to something else. Therefore you can use *get_data()* to retrieve the actual data of the string and can handle encoding yourself. Or you use get_unicode() to return a decoded UTF-16 string, which might cause problems during printing or saving. If you want a representation of the string, which should be printable in python you ca use *get()* which escapes invalid characters.

Parameters

- **buff** (BuffHandle) – a string which represents a Buff object of the string_data_item

- **cm** (*ClassManager*) – a ClassManager object

get()
Returns a MUTF8String object

get_data()
Return a series of MUTF-8 code units (a.k.a. octets, a.k.a. bytes) followed by a byte of value 0

Return type string

get_length()
Get the length of the raw string including the ULEB128 coded length and the null byte terminator

Returns int

get_obj()

get_off()

get_raw()
Returns the raw string including the ULEB128 coded length and null byte string terminator

Returns bytes

get_utf16_size()
Return the size of this string, in UTF-16 code units

:rtype:int

set_off(*off*)

show()

class androguard.core.bytecodes.dvm.**StringIdItem**(*buff*, *cm*)
Bases: object

This class can parse a string_id_item of a dex file

Parameters

- **buff** (*Buff object*) – a string which represents a Buff object of the string_id_item
- **cm** (*ClassManager*) – a ClassManager object

get_length()

get_obj()

get_off()

get_raw()

get_string_data_off()
Return the offset from the start of the file to the string data for this item

Return type int

set_off(*off*)

show()

class androguard.core.bytecodes.dvm.**TryItem**(*buff*, *cm*)
Bases: object

This class represents the try_item format

Parameters

- **buff** (BuffHandle) – a raw buffer where are the try_item format

- **cm** (ClassManager) – the ClassManager

get_handler_off()

Get the offset in bytes from the start of the associated *EncodedCatchHandlerList* to the *EncodedCatchHandler* for this entry.

Return type int

get_insn_count()

Get the number of 16-bit code units covered by this entry

Return type int

get_length()

get_off()

get_raw()

get_start_addr()

Get the start address of the block of code covered by this entry. The address is a count of 16-bit code units to the start of the first covered instruction.

Return type int

set_off(*off*)

class androguard.core.bytecodes.dvm.**TypeHIdItem**(*size*, *buff*, *cm*)

Bases: object

This class can parse a list of type_id_item of a dex file

Parameters

- **buff** (*Buff object*) – a string which represents a Buff object of the list of type_id_item
- **cm** (*ClassManager*) – a ClassManager object

get(*idx*)

get_length()

get_obj()

get_off()

get_raw()

get_type()

Return the list of type_id_item

Return type a list of *TypeIdItem* objects

set_off(*off*)

show()

class androguard.core.bytecodes.dvm.**TypeIdItem**(*buff*, *cm*)

Bases: object

This class can parse a type_id_item of a dex file

Parameters

- **buff** (*Buff object*) – a string which represents a Buff object of the type_id_item
- **cm** (*ClassManager*) – a ClassManager object

get_descriptor_idx()

Return the index into the string_ids list for the descriptor string of this type

Return type int

get_descriptor_idx_value()

Return the string associated to the descriptor

Return type string

get_length()

get_obj()

get_raw()

show()

class androguard.core.bytecodes.dvm.**TypeItem**(*buff*, *cm*)

Bases: object

This class can parse a type_item of a dex file

Parameters

- **buff** (*Buff object*) – a string which represents a Buff object of the type_item
- **cm** (*ClassManager*) – a ClassManager object

get_length()

get_obj()

get_raw()

get_string()

Return the type string

Return type string

get_type_idx()

Return the index into the type_ids list

Return type int

show()

class androguard.core.bytecodes.dvm.**TypeList**(*buff*, *cm*)

Bases: object

This class can parse a type_list of a dex file

Parameters

- **buff** (*Buff object*) – a string which represents a Buff object of the type_list
- **cm** (*ClassManager*) – a ClassManager object

get_length()

get_list()

Return the list of TypeItem

Return type a list of *TypeItem* objects

get_obj()

get_off()

get_pad()
Return the alignment string

Return type string

get_raw()

get_size()
Return the size of the list, in entries

Return type int

get_string()
Return the concatenation of all strings

Return type string

get_type_list_off()
Return the offset of the item

Return type int

set_off(*off*)

show()

androguard.core.bytecodes.dvm.**clean_name_instruction**(*instruction*)
USED IN ELSIM

androguard.core.bytecodes.dvm.**determineException**(*vm*, *m*)
Returns try-catch handler inside the method.

Parameters

- **vm** – a *DalvikVMFormat*
- **m** – a *EncodedMethod*

Returns

androguard.core.bytecodes.dvm.**determineNext**(*i*, *cur_idx*, *m*)
Determine the next offsets inside the bytecode of an *EncodedMethod*. The offsets are calculated in number of bytes from the start of the method. Note, that offsets inside the bytecode are denoted in 16bit units but this method returns actual bytes!

Offsets inside the opcode are counted from the beginning of the opcode.

The returned type is a list, as branching opcodes will have multiple paths. *if* and *switch* opcodes will return more than one item in the list, while *throw*, *return* and *goto* opcodes will always return a list with length one.

An offset of -1 indicates that the method is exited, for example by *throw* or *return*.

If the entered opcode is not branching or jumping, an empty list is returned.

Parameters

- **i** (Instruction) – the current Instruction
- **cur_idx** (*int*) – Index of the instruction
- **m** (EncodedMethod) – the current method

Returns

Return type list

androguard.core.bytecodes.dvm.**get_access_flags_string**(*value*)
Transform an access flag field to the corresponding string

Parameters **value** (*int*) – the value of the access flags

Return type string

androguard.core.bytecodes.dvm.**get_byte**(*cm*, *buff*)

androguard.core.bytecodes.dvm.**get_bytecodes_method**(*dex_object*, *ana_object*, *method*)

androguard.core.bytecodes.dvm.**get_bytecodes_methodx**(*method*, *mx*)

androguard.core.bytecodes.dvm.**get_instruction**(*cm*, *op_value*, *buff*)

Return the *Instruction* for the given opcode

Parameters

- **cm** (ClassManager) – ClassManager to propagate to Instruction
- **op_value** (*int*) – integer value of the instruction
- **buff** (*bytearray*) – Bytecode starting with the instruction

Returns the parsed Instruction

Return type *Instruction*

androguard.core.bytecodes.dvm.**get_instruction_payload**(*op_value*, *cm*, *buff*)

androguard.core.bytecodes.dvm.**get_kind**(*cm*, *kind*, *value*)

Return the value of the 'kind' argument

Parameters

- **cm** (*ClassManager*) – a ClassManager object
- **kind** (*int*) – the type of the 'kind' argument
- **value** (*int*) – the value of the 'kind' argument

Return type string

androguard.core.bytecodes.dvm.**get_optimized_instruction**(*cm*, *op_value*, *buff*)

androguard.core.bytecodes.dvm.**get_params_info**(*nb*, *proto*)

androguard.core.bytecodes.dvm.**get_sbyte**(*cm*, *buff*)

androguard.core.bytecodes.dvm.**get_type**(*atype*, *size=None*)

Retrieve the type of a descriptor (e.g : I)

androguard.core.bytecodes.dvm.**read_null_terminated_string**(*f*)

Read a null terminated string from a file-like object. :param f: file-like object :rtype: bytearray

androguard.core.bytecodes.dvm.**readsleb128**(*cm*, *buff*)

Read a signed LEB128 at the current position of the buffer.

Parameters **buff** – a file like object

Returns decoded sLEB128

androguard.core.bytecodes.dvm.**readuleb128**(*cm*, *buff*)

Read an unsigned LEB128 at the current position of the buffer

Parameters **buff** – a file like object

Returns decoded unsigned LEB128

androguard.core.bytecodes.dvm.**readuleb128p1**(*cm*, *buff*)

Read an unsigned LEB128p1 at the current position of the buffer. This format is the same as uLEB128 but has the ability to store the value -1.

Parameters **`buff`** – a file like object

Returns decoded uLEB128p1

`androguard.core.bytecodes.dvm.`**`static_operand_instruction`**(*instruction*)
USED IN ELSIM

`androguard.core.bytecodes.dvm.`**`writesleb128`**(*cm*, *value*)
Convert an integer value to the corresponding signed LEB128

Parameters **`value`** – integer value

Returns bytes

`androguard.core.bytecodes.dvm.`**`writeuleb128`**(*cm*, *value*)
Convert an integer value to the corresponding unsigned LEB128.

Raises a value error, if the given value is negative.

Parameters **`value`** – non-negative integer

Returns bytes

androguard.core.bytecodes.axml module

`class` `androguard.core.bytecodes.axml.`**`ARSCComplex`**(*buff*, *parent=None*)
Bases: `object`

This is actually a *ResTable_map_entry*

It contains a set of {name: value} mappings, which are of type *ResTable_map*. A *ResTable_map* contains two items: *ResTable_ref* and *Res_value*.

See http://androidxref.com/9.0.0_r3/xref/frameworks/base/libs/androidfw/include/androidfw/ResourceTypes.h#1485 for *ResTable_map_entry* and http://androidxref.com/9.0.0_r3/xref/frameworks/base/libs/androidfw/include/androidfw/ResourceTypes.h#1498 for *ResTable_map*

`class` `androguard.core.bytecodes.axml.`**`ARSCHeader`**(*buff*, *expected_type=None*)
Bases: `object`

Object which contains a Resource Chunk. This is an implementation of the *ResChunk_header*.

It will throw an `ResParserError` if the header could not be read successfully.

It is not checked if the data is outside the buffer size nor if the current chunk fits into the parent chunk (if any)!

The parameter *expected_type* can be used to immediately check the header for the type or raise a `ResParserError`. This is useful if you know what type of chunk must follow.

See http://androidxref.com/9.0.0_r3/xref/frameworks/base/libs/androidfw/include/androidfw/ResourceTypes.h#196 :raises: ResParserError

`SIZE = 8`

`property end`
Get the absolute offset inside the file, where the chunk ends. This is equal to *ARSCHeader.start + ARSCHeader.size*.

`property header_size`
Size of the chunk header (in bytes). Adding this value to the address of the chunk allows you to find its associated data (if any).

property size
Total size of this chunk (in bytes). This is the chunkSize plus the size of any data associated with the chunk. Adding this value to the chunk allows you to completely skip its contents (including any child chunks). If this value is the same as chunkSize, there is no data associated with the chunk.

property type
Type identifier for this chunk

class `androguard.core.bytecodes.axml.`**ARSCParser**(*raw_buff*)
Bases: `object`

Parser for resource.arsc files

The ARSC File is, like the binary XML format, a chunk based format. Both formats are actually identical but use different chunks in order to store the data.

The most outer chunk in the ARSC file is a chunk of type RES_TABLE_TYPE. Inside this chunk is a StringPool and at least one package.

Each package is a chunk of type RES_TABLE_PACKAGE_TYPE. It contains again many more chunks.

class ResourceResolver(*android_resources*, *config=None*)
Bases: `object`

Resolves resources by ID and configuration. This resolver deals with complex resources as well as with references.

put_ate_value(*result*, *ate*, *config*)
Put a ResTableEntry into the list of results :param list result: results array :param ARSCResTableEntry ate: :param ARSCResTableConfig config: :return:

put_item_value(*result*, *item*, *config*, *parent*, *complex_*)
Put the tuple (ARSCResTableConfig, resolved string) into the result set

Parameters

- **result** (`list`) – the result set
- **item** (`ARSCResStringPoolRef`) –
- **config** (`ARSCResTableConfig`) –
- **parent** (`ARSCResTableEntry`) – the originating entry
- **complex** (`bool`) – True if the originating *`ARSCResTableEntry`* was complex

Returns

resolve(*res_id*)
the given ID into the Resource and returns a list of matching resources.

Parameters **res_id** (`int`) – numerical ID of the resource

Returns a list of tuples of (ARSCResTableConfig, str)

get_bool_resources(*package_name*, *locale='\x00\x00'*)
Get the XML (as string) of all resources of type 'bool'.

Read more about bool resources: https://developer.android.com/guide/topics/resources/more-resources.html#Bool

Parameters

- **package_name** – the package name to get the resources for
- **locale** – the locale to get the resources for (default: '')

get_color_resources(*package_name*, *locale='\x00\x00'*)
Get the XML (as string) of all resources of type 'color'.

Read more about color resources: https://developer.android.com/guide/topics/resources/more-resources.html#Color

Parameters

- **`package_name`** – the package name to get the resources for
- **`locale`** – the locale to get the resources for (default: '')

`get_dimen_resources`(*package_name*, *locale='\x00\x00'*)

Get the XML (as string) of all resources of type 'dimen'.

Read more about Dimension resources: https://developer.android.com/guide/topics/resources/more-resources.html#Dimension

Parameters

- **`package_name`** – the package name to get the resources for
- **`locale`** – the locale to get the resources for (default: '')

`get_id`(*package_name*, *rid*, *locale='\x00\x00'*)

Returns the tuple (resource_type, resource_name, resource_id) for the given resource_id.

Parameters

- **`package_name`** – package name to query
- **`rid`** – the resource_id
- **`locale`** – specific locale

Returns tuple of (resource_type, resource_name, resource_id)

`get_id_resources`(*package_name*, *locale='\x00\x00'*)

Get the XML (as string) of all resources of type 'id'.

Read more about ID resources: https://developer.android.com/guide/topics/resources/more-resources.html#Id

Parameters

- **`package_name`** – the package name to get the resources for
- **`locale`** – the locale to get the resources for (default: '')

`get_integer_resources`(*package_name*, *locale='\x00\x00'*)

Get the XML (as string) of all resources of type 'integer'.

Read more about integer resources: https://developer.android.com/guide/topics/resources/more-resources.html#Integer

Parameters

- **`package_name`** – the package name to get the resources for
- **`locale`** – the locale to get the resources for (default: '')

`get_items`(*package_name*)

`get_locales`(*package_name*)

Retrieve a list of all available locales in a given packagename.

Parameters **`package_name`** – the package name to get locales of

`get_packages_names`()

Retrieve a list of all package names, which are available in the given resources.arsc.

`get_public_resources`(*package_name*, *locale='\x00\x00'*)

Get the XML (as string) of all resources of type 'public'.

The public resources table contains the IDs for each item.

Parameters

- **`package_name`** – the package name to get the resources for
- **`locale`** – the locale to get the resources for (default: '')

`get_res_configs`(*rid*, *config=None*, *fallback=True*)

Return the resources found with the ID *rid* and select the right one based on the configuration, or return all if no configuration was set.

But we try to be generous here and at least try to resolve something: This method uses a fallback to return at least one resource (the first one in the list) if more than one items are found and the default config is used and no default entry could be found.

This is usually a bad sign (i.e. the developer did not follow the android documentation: https://developer.android.com/guide/topics/resources/localization.html#failing2) In practise an app might just be designed to run on a single locale and thus only has those locales set.

You can disable this fallback behaviour, to just return exactly the given result.

Parameters

- **`rid`** – resource id as int
- **`config`** – a config to resolve from, or None to get all results
- **`fallback`** – Enable the fallback for resolving default configuration (default: True)

Returns a list of ARSCResTableConfig: ARSCResTableEntry

`get_res_id_by_key`(*package_name*, *resource_type*, *key*)

`get_resolved_res_configs`(*rid*, *config=None*)

Return a list of resolved resource IDs with their corresponding configuration. It has a similar return type as *`get_res_configs()`* but also handles complex entries and references. Also instead of returning *`ARSCResTableEntry`* in the tuple, the actual values are resolved.

This is the preferred way of resolving resource IDs to their resources.

Parameters

- **`rid`** (*`int`*) – the numerical ID of the resource
- **`config`** (*`ARSCTableResConfig`*) – the desired configuration or None to retrieve all

Returns A list of tuples of (ARSCResTableConfig, str)

`get_resolved_strings`()

`get_resource_bool`(*ate*)

`get_resource_color`(*ate*)

`get_resource_dimen`(*ate*)

`get_resource_id`(*ate*)

`get_resource_integer`(*ate*)

`get_resource_string`(*ate*)

`get_resource_style`(*ate*)

`get_resource_xml_name`(*r_id*, *package=None*)

Returns the XML name for a resource, including the package name if package is None. A full name might look like *@com.example:string/foobar* Otherwise the name is only looked up in the specified package and is returned without the package name. The same example from about without the package name will read as *@string/foobar*.

If the ID could not be found, *None* is returned.

A description of the XML name can be found here: https://developer.android.com/guide/topics/resources/providing-resources#ResourcesFromXml

Parameters

- **r_id** – numerical ID if the resource
- **package** – package name

Returns XML name identifier

get_string(*package_name*, *name*, *locale='\x00\x00'*)

get_string_resources(*package_name*, *locale='\x00\x00'*)
Get the XML (as string) of all resources of type 'string'.

Read more about string resources: https://developer.android.com/guide/topics/resources/string-resource.html

Parameters

- **package_name** – the package name to get the resources for
- **locale** – the locale to get the resources for (default: '')

get_strings_resources()
Get the XML (as string) of all resources of type 'string'. This is a combined variant, which has all locales and all package names stored.

get_type_configs(*package_name*, *type_name=None*)

get_types(*package_name*, *locale='\x00\x00'*)
Retrieve a list of all types which are available in the given package and locale.

Parameters

- **package_name** – the package name to get types of
- **locale** – the locale to get types of (default: '')

static parse_id(*name*)
Resolves an id from a binary XML file in the form "@[package:]DEADBEEF" and returns a tuple of package name and resource id. If no package name was given, i.e. the ID has the form "@DEADBEEF", the package name is set to None.

Raises a ValueError if the id is malformed.

Parameters **name** – the string of the resource, as in the binary XML file

Returns a tuple of (resource_id, package_name).

class androguard.core.bytecodes.axml.**ARSCResStringPoolRef**(*buff*, *parent=None*)
Bases: object

This is actually a *Res_value* It holds information about the stored resource value

See: http://androidxref.com/9.0.0_r3/xref/frameworks/base/libs/androidfw/include/androidfw/ResourceTypes.h#262

format_value()
Return the formatted (interpreted) data according to *data_type*.

get_data()

get_data_type()

get_data_type_string()

get_data_value()

is_reference()

Returns True if the Res_value is actually a reference to another resource

class androguard.core.bytecodes.axml.**ARSCResTableConfig**(*buff=None*, ***kwargs*)

Bases: object

ARSCResTableConfig contains the configuration for specific resource selection. This is used on the device to determine which resources should be loaded based on different properties of the device like locale or displaysize.

See the definition of *ResTable_config* in http://androidxref.com/9.0.0_r3/xref/frameworks/base/libs/androidfw/include/androidfw/ResourceTypes.h#911

classmethod default_config()

get_config_name_friendly()

Here for legacy reasons.

use *get_qualifier()* instead.

get_country()

get_density()

get_language()

get_language_and_region()

Returns the combined language+region string or for the default locale :return:

get_qualifier()

Return resource name qualifier for the current configuration. for example * *ldpi-v4* * *hdpi-v4*

All possible qualifiers are listed in table 2 of https://developer.android.com/guide/topics/resources/providing-resources

..todo:: This name might not have all properties set! Therefore returned values might not reflect the true qualifier name! :return: str

is_default()

Test if this is a default resource, which matches all

This is indicated that all fields are zero. :return: True if default, False otherwise

class androguard.core.bytecodes.axml.**ARSCResTableEntry**(*buff*, *mResId*, *parent=None*)

Bases: object

A *ResTable_entry*.

See http://androidxref.com/9.0.0_r3/xref/frameworks/base/libs/androidfw/include/androidfw/ResourceTypes.h#1458

FLAG_COMPLEX = 1

FLAG_PUBLIC = 2

FLAG_WEAK = 4

get_index()

get_key_data()

get_value()

is_complex()

is_public()

is_weak()

class androguard.core.bytecodes.axml.**ARSCResTablePackage**(*buff*, *header*)

Bases: `object`

A *ResTable_package*

See http://androidxref.com/9.0.0_r3/xref/frameworks/base/libs/androidfw/include/androidfw/ResourceTypes.h#861

get_name()

class androguard.core.bytecodes.axml.**ARSCResType**(*buff*, *parent=None*)

Bases: `object`

This is a *ResTable_type* without it's *ResChunk_header*. It contains a *ResTable_config*

See http://androidxref.com/9.0.0_r3/xref/frameworks/base/libs/androidfw/include/androidfw/ResourceTypes.h#1364

get_package_name()

get_type()

class androguard.core.bytecodes.axml.**ARSCResTypeSpec**(*buff*, *parent=None*)

Bases: `object`

See http://androidxref.com/9.0.0_r3/xref/frameworks/base/libs/androidfw/include/androidfw/ResourceTypes.h#1327

class androguard.core.bytecodes.axml.**AXMLParser**(*raw_buff*)

Bases: `object`

AXMLParser reads through all chunks in the AXML file and implements a state machine to return information about the current chunk, which can then be read by *AXMLPrinter*.

An AXML file is a file which contains multiple chunks of data, defined by the *ResChunk_header*. There is no real file magic but as the size of the first header is fixed and the *type* of the *ResChunk_header* is set to *RES_XML_TYPE*, a file will usually start with *0x03000800*. But there are several examples where the *type* is set to something else, probably in order to fool parsers.

Typically the AXMLParser is used in a loop which terminates if *m_event* is set to *END_DOCUMENT*. You can use the *next()* function to get the next chunk. Note that not all chunk types are yielded from the iterator! Some chunks are processed in the AXMLParser only. The parser will set *is_valid()* to False if it parses something not valid. Messages what is wrong are logged.

See http://androidxref.com/9.0.0_r3/xref/frameworks/base/libs/androidfw/include/androidfw/ResourceTypes.h#563

property comment

Return the comment at the current position or None if no comment is given

This works only for Tags, as the comments of Namespaces are silently dropped. Currently, there is no way of retrieving comments of namespaces.

getAttributeCount()

Return the number of Attributes for a Tag or -1 if not in a tag

getAttributeName(*index*)
Returns the String which represents the attribute name

getAttributeNamespace(*index*)
Return the Namespace URI (if any) for the attribute

getAttributeUri(*index*)
Returns the numeric ID for the namespace URI of an attribute

getAttributeValue(*index*)
This function is only used to look up strings All other work is done by `format_value()` # FIXME should unite those functions :param index: index of the attribute :return:

getAttributeValueData(*index*)
Return the data of the attribute at the given index

Parameters **index** – index of the attribute

getAttributeValueType(*index*)
Return the type of the attribute at the given index

Parameters **index** – index of the attribute

getName()
Legacy only! use `name` instead

getPrefix()
Legacy only! use `namespace` instead

getText()
Legacy only! use `text` instead

is_valid()
Get the state of the AXMLPrinter. if an error happend somewhere in the process of parsing the file, this flag is set to False.

property name
Return the String assosciated with the tag name

property namespace
Return the Namespace URI (if any) as a String for the current tag

property nsmap
Returns the current namespace mapping as a dictionary

there are several problems with the map and we try to guess a few things here:

1) a URI can be mapped by many prefixes, so it is to decide which one to take

2) a prefix might map to an empty string (some packers)

3) uri+prefix mappings might be included several times

4) prefix might be empty

property text
Return the String assosicated with the current text

class androguard.core.bytecodes.axml.**AXMLPrinter**(*raw_buff*)
Bases: `object`

Converter for AXML Files into a lxml ElementTree, which can easily be converted into XML.

A Reference Implementation can be found at http://androidxref.com/9.0.0_r3/xref/frameworks/base/tools/aapt/XMLNode.cpp

get_buff()
Returns the raw XML file without prettification applied.

Returns bytes, encoded as UTF-8

get_xml(*pretty=True*)
Get the XML as an UTF-8 string

Returns bytes encoded as UTF-8

get_xml_obj()
Get the XML as an ElementTree object

Returns `lxml.etree.Element`

is_packed()
Returns True if the AXML is likely to be packed

Packers do some weird stuff and we try to detect it. Sometimes the files are not packed but simply broken or compiled with some broken version of a tool. Some file corruption might also be appear to be a packed file.

Returns True if packer detected, False otherwise

is_valid()
Return the state of the AXMLParser. If this flag is set to False, the parsing has failed, thus the resulting XML will not work or will even be empty.

class androguard.core.bytecodes.axml.**PackageContext**(*current_package*, *stringpool_main*, *mTableStrings*, *mKeyStrings*)
Bases: `object`

get_mResId()

get_package_name()

set_mResId(*mResId*)

exception androguard.core.bytecodes.axml.**ResParserError**
Bases: `Exception`

Exception for the parsers

class androguard.core.bytecodes.axml.**StringBlock**(*buff*, *header*)
Bases: `object`

StringBlock is a CHUNK inside an AXML File: *ResStringPool_header* It contains all strings, which are used by referecing to ID's

See http://androidxref.com/9.0.0_r3/xref/frameworks/base/libs/androidfw/include/androidfw/ResourceTypes.h#436

getString(*idx*)
Return the string at the index in the string table

Parameters **idx** – index in the string table

Returns str

getStyle(*idx*)
Return the style associated with the index

Parameters **idx** – index of the style

Returns

show()
Print some information on stdout about the string table

androguard.core.bytecodes.axml.**complexToFloat**(*xcomplex*)
Convert a complex unit into float

androguard.core.bytecodes.axml.**format_value**(*_type*, *_data*, *lookup_string=<function <lambda>>*)
Format a value based on type and data. By default, no strings are looked up and "<string>" is returned. You need to define *lookup_string* in order to actually lookup strings from the string table.

Parameters

- **_type** – The numeric type of the value
- **_data** – The numeric data of the value
- **lookup_string** – A function how to resolve strings from integer IDs

androguard.core.bytecodes.axml.**get_arsc_info**(*arscobj*)
Return a string containing all resources packages ordered by packagename, locale and type.

Parameters **arscobj** – *ARSCParser*

Returns a string

androguard.core.mutf8 module

class androguard.core.mutf8.**MUTF8String**(*b*)
Bases: bytes

find(*sub*[, *start*[, *end*]]) → int
Return the lowest index in B where subsection sub is found, such that sub is contained within B[start,end]. Optional arguments start and end are interpreted as in slice notation.

Return -1 on failure.

classmethod from_str(*s*)

classmethod join(*data*, *spacing=b"*)
Concatenate any number of bytes objects.

The bytes whose method is called is inserted in between each pair.

The result is returned as a new bytes object.

Example: b'.'.join([b'ab', b'pq', b'rs']) -> b'ab.pq.rs'.

lstrip(*sub*)
Strip leading bytes contained in the argument.

If the argument is omitted or None, strip leading ASCII whitespace.

replace(*old*, *new*, *count=None*)
Return a copy with all occurrences of substring old replaced by new.

count Maximum number of occurrences to replace. -1 (the default value) means replace all occurrences.

If the optional argument count is given, only the first count occurrences are replaced.

rsplit(*sep=None*, *maxsplit=-1*)
Return a list of the sections in the bytes, using sep as the delimiter.

sep The delimiter according which to split the bytes. None (the default value) means split on ASCII whitespace characters (space, tab, return, newline, formfeed, vertical tab).

maxsplit Maximum number of splits to do. -1 (the default value) means no limit.

Splitting is done starting at the end of the bytes and working to the front.

split(*sep=None*, *maxsplit=-1*)

Return a list of the sections in the bytes, using sep as the delimiter.

sep The delimiter according which to split the bytes. None (the default value) means split on ASCII whitespace characters (space, tab, return, newline, formfeed, vertical tab).

maxsplit Maximum number of splits to do. -1 (the default value) means no limit.

startswith(*prefix*[, *start*[, *end*]]) → bool

Return True if B starts with the specified prefix, False otherwise. With optional start, test B beginning at that position. With optional end, stop comparing B at that position. prefix can also be a tuple of bytes to try.

androguard.core.mutf8.**decode**(*b*)

androguard.core.mutf8.**encode**(*s*)

Module contents

androguard.core.resources package

Submodules

androguard.core.resources.public module

Module contents

Submodules

androguard.core.androconf module

class androguard.core.androconf.**Color**

Bases: object

Black = '\x1b[30m'

Blue = '\x1b[34m'

Bold = '\x1b[1m'

Cyan = '\x1b[36m'

Green = '\x1b[32m'

Grey = '\x1b[37m'

Normal = '\x1b[0m'

Purple = '\x1b[35m'

Red = '\x1b[31m'

Yellow = '\x1b[33m'

class androguard.core.androconf.**Configuration**

Bases: `object`

instance = {'BIN_DED': 'ded.sh', 'BIN_DEX2JAR': 'dex2jar.sh', 'BIN_FERNFLOWER': 'fernf

exception androguard.core.androconf.**InvalidResourceError**

Bases: `Exception`

Invalid Resource Erorr is thrown by load_api_specific_resource_module

androguard.core.androconf.**color_range**(*startcolor*, *goalcolor*, *steps*)

wrapper for interpolate_tuple that accepts colors as html ("#CCCCC" and such)

androguard.core.androconf.**default_colors**(*obj*)

androguard.core.androconf.**disable_colors**()

Disable colors from the output (color = normal)

androguard.core.androconf.**enable_colors**(*colors*)

androguard.core.androconf.**interpolate_tuple**(*startcolor*, *goalcolor*, *steps*)

Take two RGB color sets and mix them over a specified number of steps. Return the list

androguard.core.androconf.**is_android**(*filename*)

Return the type of the file

:param filename : the filename :returns: "APK", "DEX", None

androguard.core.androconf.**is_android_raw**(*raw*)

Returns a string that describes the type of file, for common Android specific formats

androguard.core.androconf.**is_ascii_problem**(*s*)

Test if a string contains other chars than ASCII

Parameters **s** (*MUTF8Strin*) – a string to test

Returns True if string contains other chars than ASCII, False otherwise

androguard.core.androconf.**load_api_specific_resource_module**(*resource_name*, *api=None*)

Load the module from the JSON files and return a dict, which might be empty if the resource could not be loaded.

If no api version is given, the default one from the CONF dict is used.

Parameters

- **resource_name** – Name of the resource to load
- **api** – API version

Returns dict

androguard.core.androconf.**make_color_tuple**(*color*)

turn something like "#000000" into 0,0,0 or "#FFFFFF into "255,255,255"

androguard.core.androconf.**remove_colors**()

Remove colors from the output (no escape sequences)

androguard.core.androconf.**rrmdir**(*directory*)

Recursivly delete a directory

Parameters **directory** – directory to remove

androguard.core.androconf.**save_colors**()

androguard.core.androconf.**set_options**(*key*, *value*)

Deprecated since version 3.3.5: Use `CONF[key] = value` instead

androguard.core.androconf.**show_logging**(*level=20*)

enable log messages on stdout

We will catch all messages here! From all loggers…

androguard.core.bytecode module

class androguard.core.bytecode.**Buff**(*offset*, *buff*)

Bases: `object`

class androguard.core.bytecode.**BuffHandle**(*buff*)

Bases: `object`

BuffHandle is a wrapper around bytes. It gives the ability to jump in the byte stream, just like with BytesIO.

add_idx(*idx*)

Advance the current offset by *idx*

Parameters **idx** (*int*) – number of bytes to advance

end()

Test if the current offset is at the end or over the buffer boundary

Return type bool

get_buff()

Return the whole buffer

Return type bytearray

get_idx()

Get the current offset in the buffer

Return type int

length_buff()

Alias for `size()`

peek(*size*)

Alias for `read_b()`

read(*size*)

Read from the current offset a total number of *size* bytes and increment the offset by *size*

Parameters **size** (*int*) – length of bytes to read

Return type bytearray

readNullString(*size*)

Read a String with length *size* at the current offset

Parameters **size** (*int*) – length of the string

Return type bytearray

read_at(*offset*, *size*)

Read bytes from the given offset with length *size* without incrementing the current offset

Parameters

- **offset** (*int*) – offset to start reading

- **size** (*int*) – length of bytes to read

Return type bytearray

read_b (*size*)

Read bytes with length *size* without incrementing the current offset

Parameters **size** (*int*) – length to read in bytes

Return type bytearray

readat (*off*)

Read all bytes from the start of *off* until the end of the buffer

This method can be used to determine a checksum of a buffer from a given point on.

Parameters **off** (*int*) – starting offset

Return type bytearray

save (*filename*)

Save the current buffer to *filename*

Exisiting files with the same name will be overwritten.

Parameters **filename** (*str*) – the name of the file to save to

set_buff (*buff*)

Overwrite the current buffer with the content of *buff*

Parameters **buff** (*bytearray*) – the new buffer

set_idx (*idx*)

Set the current offset in the buffer

Parameters **idx** (*int*) – offset to set

size ()

Get the total size of the buffer

Return type int

tell ()

Alias for *get_idx()*.

Return type int

androguard.core.bytecode.**FormatClassToJava** (*i*)

Transform a java class name into the typed variant found in DEX files.

example:

```
>>> FormatClassToJava('java.lang.Object')
'Ljava/lang/Object;'
```

Parameters **i** – the input class name

Return type str

androguard.core.bytecode.**FormatClassToPython** (*i*)

Transform a typed class name into a form which can be used as a python attribute

example:

```
>>> FormatClassToPython('Lfoo/bar/foo/Barfoo$InnerClass;')
'Lfoo_bar_foo_Barfoo_InnerClass'
```

Parameters `i` – classname to transform

Return type str

`androguard.core.bytecode.`**`FormatDescriptorToPython`**`(`*i*`)`

Format a descriptor into a form which can be used as a python attribute

example:

```
>>> FormatDescriptorToPython('(Ljava/lang/Long; Ljava/lang/Long; Z Z)V')
'Ljava_lang_LongLjava_lang_LongZZV
```

Parameters `i` – name to transform

Return type str

`androguard.core.bytecode.`**`FormatNameToPython`**`(`*i*`)`

Transform a (method) name into a form which can be used as a python attribute

example:

```
>>> FormatNameToPython('<clinit>')
'clinit'
```

Parameters `i` – name to transform

Return type str

`class` `androguard.core.bytecode.`**`Node`**`(`*n*, *s*`)`

Bases: `object`

`androguard.core.bytecode.`**`PrettyShow`**`(`*m_a*, *basic_blocks*, *notes={}*`)`

`androguard.core.bytecode.`**`PrettyShowEx`**`(`*exceptions*`)`

`class` `androguard.core.bytecode.`**`TmpBlock`**`(`*name*`)`

Bases: `object`

`get_name``()`

`androguard.core.bytecode.`**`disable_print_colors`**`()`

`androguard.core.bytecode.`**`enable_print_colors`**`(`*colors*`)`

`androguard.core.bytecode.`**`get_package_class_name`**`(`*name*`)`

Return package and class name in a java variant from a typed variant name.

If no package could be found, the package is an empty string.

If the name is an array type, the array is discarded.

example:

```
>>> get_package_class_name('Ljava/lang/Object;')
('java.lang', 'Object')
>>> get_package_class_name('[[Ljava/lang/Object;')
('java.lang', 'Object')
```

(continues on next page)

(continued from previous page)

```
>>> get_package_class_name('LSomeClass;')
('', 'SomeClass')
```

Parameters **name** – the name

Return type tuple

Returns

androguard.core.bytecode.**method2dot**(*mx*, *colors=None*)

Export analysis method to dot format.

A control flow graph is created by using the concept of BasicBlocks. Each BasicBlock is a sequence of opcode without any jumps or branch.

Parameters

- **mx** – *MethodAnalysis*
- **colors** – dict of colors to use, if colors is None the default colors are used

Returns a string which contains the dot graph

androguard.core.bytecode.**method2format**(*output*, *_format='png'*, *mx=None*, *raw=None*)

Export method structure as a graph to a specific file format using dot from the graphviz package. The result is written to the file specified via `output`.

There are two possibilites to give input for this method:

1) use `raw` argument and pass a dictionary containing the keys `name`, `nodes` and `edges`. This can be created using *method2dot()*. 2) give a *MethodAnalysis*.

This function requires pydot!

There is a special format `raw` which saves the dot buffer before it is handled by pydot.

Parameters

- **output** (*str*) – output filename
- **_format** (*str*) – format type (png, jpg …). Can use all formats which are understood by pydot.
- **mx** (androguard.core.analysis.analysis.MethodAnalysis) – specify the MethodAnalysis object
- **raw** (*dict*) – use directly a dot raw buffer if None

androguard.core.bytecode.**method2jpg**(*output*, *mx*, *raw=False*)

Export method to a jpg file format

Parameters

- **output** (*string*) – output filename
- **mx** (MethodAnalysis object) – specify the MethodAnalysis object
- **raw** (*string*) – use directly a dot raw buffer (optional)

androguard.core.bytecode.**method2json**(*mx*, *directed_graph=False*)

Create directed or undirected graph in the json format.

Parameters

- **mx** – *MethodAnalysis*

- **directed_graph** – True if a directed graph should be created (default: False)

Returns

androguard.core.bytecode.**method2json_direct**(*mx*)

Parameters **mx** – *MethodAnalysis*

Returns

androguard.core.bytecode.**method2json_undirect**(*mx*)

Parameters **mx** – *MethodAnalysis*

Returns

androguard.core.bytecode.**method2png**(*output*, *mx*, *raw=False*)

Export method to a png file format

Parameters

- **output** (*string*) – output filename
- **mx** (MethodAnalysis object) – specify the MethodAnalysis object
- **raw** (*string*) – use directly a dot raw buffer

androguard.core.bytecode.**object_to_bytes**(*obj*)

Convert a object to a bytearray or call get_raw() of the object if no useful type was found.

androguard.core.bytecode.**vm2json**(*vm*)

Get a JSON representation of a DEX file

Parameters **vm** – *DalvikVMFormat*

Returns

Module contents

androguard.decompiler package

Subpackages

androguard.decompiler.dad package

Submodules

androguard.decompiler.dad.dast module

This file is a simplified version of writer.py that outputs an AST instead of source code.

class androguard.decompiler.dad.dast.**JSONWriter**(*graph*, *method*)

Bases: object

add(*val*)

get_ast()

get_cond(*node*)

visit_cond_node(*cond*)

visit_ins(*op*)

visit_loop_node(*loop*)

visit_node(*node*)

visit_return_node(*ret*)

visit_statement_node(*stmt*)

visit_switch_node(*switch*)

visit_throw_node(*throw*)

visit_try_node(*try_node*)

androguard.decompiler.dad.dast.**array_access**(*arr*, *ind*)

androguard.decompiler.dad.dast.**array_creation**(*tn*, *params*, *dim*)

androguard.decompiler.dad.dast.**array_initializer**(*params*, *tn=None*)

androguard.decompiler.dad.dast.**assignment**(*lhs*, *rhs*, *op=”*)

androguard.decompiler.dad.dast.**binary_infix**(*op*, *left*, *right*)

androguard.decompiler.dad.dast.**cast**(*tn*, *arg*)

androguard.decompiler.dad.dast.**dummy**(**args*)

androguard.decompiler.dad.dast.**expression_stmt**(*expr*)

androguard.decompiler.dad.dast.**field_access**(*triple*, *left*)

androguard.decompiler.dad.dast.**if_stmt**(*cond_expr*, *scopes*)

androguard.decompiler.dad.dast.**jump_stmt**(*keyword*)

androguard.decompiler.dad.dast.**literal**(*result*, *tt*)

androguard.decompiler.dad.dast.**literal_bool**(*b*)

androguard.decompiler.dad.dast.**literal_class**(*desc*)

androguard.decompiler.dad.dast.**literal_double**(*f*)

androguard.decompiler.dad.dast.**literal_float**(*f*)

androguard.decompiler.dad.dast.**literal_hex_int**(*b*)

androguard.decompiler.dad.dast.**literal_int**(*b*)

androguard.decompiler.dad.dast.**literal_long**(*b*)

androguard.decompiler.dad.dast.**literal_null**()

androguard.decompiler.dad.dast.**literal_string**(*s*)

androguard.decompiler.dad.dast.**local**(*name*)

androguard.decompiler.dad.dast.**local_decl_stmt**(*expr*, *decl*)

androguard.decompiler.dad.dast.**loop_stmt**(*isdo*, *cond_expr*, *body*)

androguard.decompiler.dad.dast.**method_invocation**(*triple*, *name*, *base*, *params*)

androguard.decompiler.dad.dast.**parenthesis**(*expr*)

androguard.decompiler.dad.dast.**parse_descriptor**(*desc*)

androguard.decompiler.dad.dast.**return_stmt**(*expr*)

androguard.decompiler.dad.dast.**statement_block**()

androguard.decompiler.dad.dast.**switch_stmt**(*cond_expr*, *ksv_pairs*)

androguard.decompiler.dad.dast.**throw_stmt**(*expr*)

androguard.decompiler.dad.dast.**try_stmt**(*tryb*, *pairs*)

androguard.decompiler.dad.dast.**typen**(*baset*, *dim*)

androguard.decompiler.dad.dast.**unary_postfix**(*left*, *op*)

androguard.decompiler.dad.dast.**unary_prefix**(*op*, *left*)

androguard.decompiler.dad.dast.**var_decl**(*typen*, *var*)

androguard.decompiler.dad.dast.**visit_arr_data**(*value*)

androguard.decompiler.dad.dast.**visit_decl**(*var*, *init_expr=None*)

androguard.decompiler.dad.dast.**visit_expr**(*op*)

androguard.decompiler.dad.dast.**visit_ins**(*op*, *isCtor=False*)

androguard.decompiler.dad.dast.**write_inplace_if_possible**(*lhs*, *rhs*)

androguard.decompiler.dad.basic_blocks module

class androguard.decompiler.dad.basic_blocks.**BasicBlock**(*name*, *block_ins*)

Bases: *androguard.decompiler.dad.node.Node*

add_ins(*new_ins_list*)

add_variable_declaration(*variable*)

get_ins()

get_loc_with_ins()

number_ins(*num*)

remove_ins(*loc*, *ins*)

set_catch_type(*_type*)

class androguard.decompiler.dad.basic_blocks.**CatchBlock**(*node*)

Bases: *androguard.decompiler.dad.basic_blocks.BasicBlock*

visit(*visitor*)

visit_exception(*visitor*)

class androguard.decompiler.dad.basic_blocks.**CondBlock**(*name*, *block_ins*)

Bases: *androguard.decompiler.dad.basic_blocks.BasicBlock*

neg()

update_attribute_with(*n_map*)

visit(*visitor*)

visit_cond(*visitor*)

class androguard.decompiler.dad.basic_blocks.**Condition**(*cond1*, *cond2*, *isand*, *isnot*)

Bases: object

get_ins()

get_loc_with_ins()

neg()

visit(*visitor*)

class androguard.decompiler.dad.basic_blocks.**LoopBlock**(*name*, *cond*)
Bases: *androguard.decompiler.dad.basic_blocks.CondBlock*

get_ins()

get_loc_with_ins()

neg()

update_attribute_with(*n_map*)

visit(*visitor*)

visit_cond(*visitor*)

class androguard.decompiler.dad.basic_blocks.**ReturnBlock**(*name*, *block_ins*)
Bases: *androguard.decompiler.dad.basic_blocks.BasicBlock*

visit(*visitor*)

class androguard.decompiler.dad.basic_blocks.**ShortCircuitBlock**(*name*, *cond*)
Bases: *androguard.decompiler.dad.basic_blocks.CondBlock*

get_ins()

get_loc_with_ins()

neg()

visit_cond(*visitor*)

class androguard.decompiler.dad.basic_blocks.**StatementBlock**(*name*, *block_ins*)
Bases: *androguard.decompiler.dad.basic_blocks.BasicBlock*

visit(*visitor*)

class androguard.decompiler.dad.basic_blocks.**SwitchBlock**(*name*, *switch*, *block_ins*)
Bases: *androguard.decompiler.dad.basic_blocks.BasicBlock*

add_case(*case*)

copy_from(*node*)

order_cases()

update_attribute_with(*n_map*)

visit(*visitor*)

class androguard.decompiler.dad.basic_blocks.**ThrowBlock**(*name*, *block_ins*)
Bases: *androguard.decompiler.dad.basic_blocks.BasicBlock*

visit(*visitor*)

class androguard.decompiler.dad.basic_blocks.**TryBlock**(*node*)
Bases: *androguard.decompiler.dad.basic_blocks.BasicBlock*

add_catch_node(*node*)

property num

visit(*visitor*)

androguard.decompiler.dad.basic_blocks.**build_node_from_block**(*block*, *vmap*, *gen_ret*, *exception_type=None*)

androguard.decompiler.dad.control_flow module

androguard.decompiler.dad.control_flow.**catch_struct**(*graph*, *idoms*)

androguard.decompiler.dad.control_flow.**derived_sequence**(*graph*)
Compute the derived sequence of the graph G The intervals of G are collapsed into nodes, intervals of these nodes are built, and the process is repeated iteratively until we obtain a single node (if the graph is not irreducible)

androguard.decompiler.dad.control_flow.**identify_structures**(*graph*, *idoms*)

androguard.decompiler.dad.control_flow.**if_struct**(*graph*, *idoms*)

androguard.decompiler.dad.control_flow.**intervals**(*graph*)
Compute the intervals of the graph Returns interval_graph: a graph of the intervals of G interv_heads: a dict of (header node, interval)

androguard.decompiler.dad.control_flow.**loop_follow**(*start*, *end*, *nodes_in_loop*)

androguard.decompiler.dad.control_flow.**loop_struct**(*graphs_list*, *intervals_list*)

androguard.decompiler.dad.control_flow.**loop_type**(*start*, *end*, *nodes_in_loop*)

androguard.decompiler.dad.control_flow.**mark_loop**(*graph*, *start*, *end*, *interval*)

androguard.decompiler.dad.control_flow.**mark_loop_rec**(*graph*, *node*, *s_num*, *e_num*, *interval*, *nodes_in_loop*)

androguard.decompiler.dad.control_flow.**short_circuit_struct**(*graph*, *idom*, *node_map*)

androguard.decompiler.dad.control_flow.**switch_struct**(*graph*, *idoms*)

androguard.decompiler.dad.control_flow.**update_dom**(*idoms*, *node_map*)

androguard.decompiler.dad.control_flow.**while_block_struct**(*graph*, *node_map*)

androguard.decompiler.dad.dataflow module

class androguard.decompiler.dad.dataflow.**BasicReachDef**(*graph*, *params*)
Bases: object

run()

class androguard.decompiler.dad.dataflow.**DummyNode**(*name*)
Bases: *androguard.decompiler.dad.node.Node*

get_loc_with_ins()

androguard.decompiler.dad.dataflow.**build_def_use**(*graph*, *lparams*)
Builds the Def-Use and Use-Def (DU/UD) chains of the variables of the method.

androguard.decompiler.dad.dataflow.**clear_path**(*graph*, *reg*, *loc1*, *loc2*)
Check that the path from loc1 to loc2 is clear. We have to check that there is no side effect between the two location points. We also have to check that the variable *reg* is not redefined along one of the possible pathes from loc1 to loc2.

androguard.decompiler.dad.dataflow.**clear_path_node**(*graph*, *reg*, *loc1*, *loc2*) androguard.decompiler.dad.dataflow.**dead_code_elimination**(*graph*, *du*, *ud*)

Run a dead code elimination pass. Instructions are checked to be dead. If it is the case, we remove them and we update the DU & UD chains of its variables to check for further dead instructions.

androguard.decompiler.dad.dataflow.**group_variables**(*lvars*, *DU*, *UD*)

androguard.decompiler.dad.dataflow.**place_declarations**(*graph*, *dvars*, *du*, *ud*)

androguard.decompiler.dad.dataflow.**reach_def_analysis**(*graph*, *lparams*)

androguard.decompiler.dad.dataflow.**register_propagation**(*graph*, *du*, *ud*)

Propagate the temporary registers between instructions and remove them if necessary. We process the nodes of the graph in reverse post order. For each instruction in the node, we look at the variables that it uses. For each of these variables we look where it is defined and if we can replace it with its definition. We have to be careful to the side effects some instructions may have. To do the propagation, we use the computed DU and UD chains.

androguard.decompiler.dad.dataflow.**split_variables**(*graph*, *lvars*, *DU*, *UD*)

androguard.decompiler.dad.dataflow.**update_chain**(*graph*, *loc*, *du*, *ud*)

Updates the DU chain of the instruction located at loc such that there is no more reference to it so that we can remove it. When an instruction is found to be dead (i.e it has no side effect, and the register defined is not used) we have to update the DU chain of all the variables that may me used by the dead instruction.

androguard.decompiler.dad.decompile module

class androguard.decompiler.dad.decompile.**DvClass**(*dvclass*, *vma*)

Bases: `object`

This is a wrapper for *ClassDefItem* inside the decompiler.

At first, `methods` contains a list of `EncodedMethods`, which are successively replaced by *DvMethod* in the process of decompilation.

get_ast()

get_methods()

get_source()

get_source_ext()

process(*doAST=False*)

process_method(*num*, *doAST=False*)

show_source()

class androguard.decompiler.dad.decompile.**DvMachine**(*name*)

Bases: `object`

Wrapper class for a Dalvik Object, like a DEX or ODEX file.

The wrapper allows to take a Dalvik file and get a list of Classes out of it. The *DvMachine* can take either an APK file directly, where all DEX files from the multidex are used, or a single DEX or ODEX file as an argument.

At first, `classes` contains only *ClassDefItem* as values. Then these objects are replaced by *DvClass* items successively.

get_ast()

Processes each class with AST enabled and returns a dictionary with all single ASTs Classnames as keys.

Returns an dictionary for all classes

Return type dict

get_class (*class_name*)
Return the `DvClass` with the given name

The name is partially matched against the known class names and the first result is returned. For example, the input *foobar* will match on Lfoobar/bla/foo;

Parameters **class_name** (*str*) –

Returns the class matching on the name

Return type *DvClass*

get_classes ()
Return a list of classnames contained in this machine. The format of each name is Lxxx;

Returns list of class names

process ()
Process all classes inside the machine.

This calls `process()` on each `DvClass`.

process_and_show ()
Run `process()` and `show_source()` after each other.

show_source ()
Calls *show_source* on all classes inside the machine. This prints the source to stdout.

This calls `show_source()` on each `DvClass`.

class androguard.decompiler.dad.decompile.**DvMethod** (*methanalysis*)
Bases: `object`

This is a wrapper around `MethodAnalysis` and `EncodedMethod` inside the decompiler.

get_ast ()

get_source ()

get_source_ext ()

process (*doAST=False*)

show_source ()

androguard.decompiler.dad.decompile.**get_field_ast** (*field*)

androguard.decompiler.dad.decompile.**main** ()

androguard.decompiler.dad.graph module

class androguard.decompiler.dad.graph.**GenInvokeRetName**
Bases: `object`

last ()

new ()

set_to (*ret*)

class androguard.decompiler.dad.graph.**Graph**
Bases: `object`

Stores a CFG (Control Flow Graph), which is a directed graph.

The CFG defines an entry node `entry`, a single exit node `exit`, a list of nodes `nodes` and a list of edges `edges`.

add_catch_edge(*e1*, *e2*)

add_edge(*e1*, *e2*)

add_node(*node*)
Adds the given node to the graph, without connecting it to anyhting else.

Parameters **node** (`androguard.decompiler.dad.node.Node`) – node to add

all_preds(*node*)

all_sucs(*node*)

compute_rpo()
Number the nodes in reverse post order. An RPO traversal visit as many predecessors of a node as possible before visiting the node itself.

draw(*name*, *dname*, *draw_branches=True*)
Writes the current graph as a PNG file

Parameters

- **name** (*str*) – filename (without .png)
- **dname** (*str*) – directory of the output png
- **draw_branches** –

Returns

get_ins_from_loc(*loc*)

get_node_from_loc(*loc*)

immediate_dominators()

number_ins()

post_order()
Yields the :class‘~androguard.decompiler.dad.node.Node‘s of the graph in post-order i.e we visit all the children of a node before visiting the node itself.

preds(*node*)

remove_ins(*loc*)

remove_node(*node*)
Remove the node from the graph, removes also all connections.

Parameters **node** (`androguard.decompiler.dad.node.Node`) – the node to remove

sucs(*node*)

`androguard.decompiler.dad.graph.`**bfs**(*start*)
Breadth first search

Yields all nodes found from the starting point

Parameters **start** – start node

`androguard.decompiler.dad.graph.`**construct**(*start_block*, *vmap*, *exceptions*)
Constructs a CFG

Parameters

- **start_block** (androguard.core.analysis.analysis.DVMBasicBlock) – The startpoint
- **vmap** – variable mapping
- **exceptions** – list of androguard.core.analysis.analysis.ExceptionAnalysis

Return type *Graph*

androguard.decompiler.dad.graph.**dom_lt**(*graph*)
Dominator algorithm from Lengauer-Tarjan

androguard.decompiler.dad.graph.**make_node**(*graph*, *block*, *block_to_node*, *vmap*, *gen_ret*)

androguard.decompiler.dad.graph.**simplify**(*graph*)
Simplify the CFG by merging/deleting statement nodes when possible: If statement B follows statement A and if B has no other predecessor besides A, then we can merge A and B into a new statement node. We also remove nodes which do nothing except redirecting the control flow (nodes which only contains a goto).

androguard.decompiler.dad.graph.**split_if_nodes**(*graph*)
Split IfNodes in two nodes, the first node is the header node, the second one is only composed of the jump condition.

androguard.decompiler.dad.instruction module

class androguard.decompiler.dad.instruction.**ArrayExpression**
Bases: *androguard.decompiler.dad.instruction.IRForm*

class androguard.decompiler.dad.instruction.**ArrayLengthExpression**(*array*)
Bases: *androguard.decompiler.dad.instruction.ArrayExpression*

get_type()

get_used_vars()

replace(*old*, *new*)

replace_var(*old*, *new*)

visit(*visitor*)

class androguard.decompiler.dad.instruction.**ArrayLoadExpression**(*arg*, *index*, *_type*)
Bases: *androguard.decompiler.dad.instruction.ArrayExpression*

get_type()

get_used_vars()

replace(*old*, *new*)

replace_var(*old*, *new*)

visit(*visitor*)

class androguard.decompiler.dad.instruction.**ArrayStoreInstruction**(*rhs*, *array*, *index*, *_type*)
Bases: *androguard.decompiler.dad.instruction.IRForm*

get_used_vars()

has_side_effect()

replace(*old*, *new*)

replace_var(*old*, *new*)

visit(*visitor*)

class androguard.decompiler.dad.instruction.**AssignExpression**(*lhs*, *rhs*)
Bases: *androguard.decompiler.dad.instruction.IRForm*

get_lhs()

get_rhs()

get_used_vars()

has_side_effect()

is_call()

is_propagable()

remove_defined_var()

replace(*old*, *new*)

replace_lhs(*new*)

replace_var(*old*, *new*)

visit(*visitor*)

class androguard.decompiler.dad.instruction.**BaseClass**(*name*, *descriptor=None*)
Bases: *androguard.decompiler.dad.instruction.IRForm*

is_const()

visit(*visitor*)

class androguard.decompiler.dad.instruction.**BinaryCompExpression**(*op*, *arg1*, *arg2*, *_type*)
Bases: *androguard.decompiler.dad.instruction.BinaryExpression*

visit(*visitor*)

class androguard.decompiler.dad.instruction.**BinaryExpression**(*op*, *arg1*, *arg2*, *_type*)
Bases: *androguard.decompiler.dad.instruction.IRForm*

get_used_vars()

has_side_effect()

replace(*old*, *new*)

replace_var(*old*, *new*)

visit(*visitor*)

class androguard.decompiler.dad.instruction.**BinaryExpression2Addr**(*op*, *dest*, *arg*, *_type*)
Bases: *androguard.decompiler.dad.instruction.BinaryExpression*

class androguard.decompiler.dad.instruction.**BinaryExpressionLit**(*op*, *arg1*, *arg2*)
Bases: *androguard.decompiler.dad.instruction.BinaryExpression*

class androguard.decompiler.dad.instruction.**CastExpression**(*op*, *atype*, *arg*)
Bases: *androguard.decompiler.dad.instruction.UnaryExpression*

get_type()

get_used_vars()

is_const()

visit(*visitor*)

class androguard.decompiler.dad.instruction.**CheckCastExpression**(*arg*, *_type*, *descriptor=None*)

Bases: `androguard.decompiler.dad.instruction.IRForm`

get_used_vars()

is_const()

replace(*old*, *new*)

replace_var(*old*, *new*)

visit(*visitor*)

class androguard.decompiler.dad.instruction.**ConditionalExpression**(*op*, *arg1*, *arg2*)

Bases: `androguard.decompiler.dad.instruction.IRForm`

get_lhs()

get_used_vars()

is_cond()

neg()

replace(*old*, *new*)

replace_var(*old*, *new*)

visit(*visitor*)

class androguard.decompiler.dad.instruction.**ConditionalZExpression**(*op*, *arg*)

Bases: `androguard.decompiler.dad.instruction.IRForm`

get_lhs()

get_used_vars()

is_cond()

neg()

replace(*old*, *new*)

replace_var(*old*, *new*)

visit(*visitor*)

class androguard.decompiler.dad.instruction.**Constant**(*value*, *atype*, *int_value=None*, *descriptor=None*)

Bases: `androguard.decompiler.dad.instruction.IRForm`

get_int_value()

get_type()

get_used_vars()

is_const()

visit(*visitor*)

class androguard.decompiler.dad.instruction.**FillArrayExpression**(*reg*, *value*)

Bases: *androguard.decompiler.dad.instruction.ArrayExpression*

get_rhs()

get_used_vars()

is_propagable()

replace(*old*, *new*)

replace_var(*old*, *new*)

visit(*visitor*)

class androguard.decompiler.dad.instruction.**FilledArrayExpression**(*asize*, *atype*, *args*)

Bases: *androguard.decompiler.dad.instruction.ArrayExpression*

get_used_vars()

replace(*old*, *new*)

replace_var(*old*, *new*)

visit(*visitor*)

class androguard.decompiler.dad.instruction.**IRForm**

Bases: object

get_lhs()

get_rhs()

get_type()

get_used_vars()

has_side_effect()

is_call()

is_cond()

is_const()

is_ident()

is_propagable()

remove_defined_var()

replace(*old*, *new*)

replace_lhs(*new*)

replace_var(*old*, *new*)

set_type(*_type*)

visit(*visitor*)

class androguard.decompiler.dad.instruction.**InstanceExpression**(*arg*, *klass*, *ftype*, *name*)

Bases: *androguard.decompiler.dad.instruction.IRForm*

get_type()

get_used_vars()

replace(*old, new*)

replace_var(*old, new*)

visit(*visitor*)

class androguard.decompiler.dad.instruction.**InstanceInstruction**(*rhs, lhs, klass, atype, name*)

Bases: *androguard.decompiler.dad.instruction.IRForm*

get_lhs()

get_used_vars()

has_side_effect()

replace(*old, new*)

replace_var(*old, new*)

visit(*visitor*)

class androguard.decompiler.dad.instruction.**InvokeDirectInstruction**(*clsname, name, base, rtype, ptype, args, triple*)

Bases: *androguard.decompiler.dad.instruction.InvokeInstruction*

class androguard.decompiler.dad.instruction.**InvokeInstruction**(*clsname, name, base, rtype, ptype, args, triple*)

Bases: *androguard.decompiler.dad.instruction.IRForm*

get_type()

get_used_vars()

has_side_effect()

is_call()

replace(*old, new*)

replace_var(*old, new*)

visit(*visitor*)

class androguard.decompiler.dad.instruction.**InvokeRangeInstruction**(*clsname, name, rtype, ptype, args, triple*)

Bases: *androguard.decompiler.dad.instruction.InvokeInstruction*

class androguard.decompiler.dad.instruction.**InvokeStaticInstruction**(*clsname, name, base, rtype, ptype, args, triple*)

Bases: *androguard.decompiler.dad.instruction.InvokeInstruction*

get_used_vars()

class androguard.decompiler.dad.instruction.**MonitorEnterExpression**(*ref*)

Bases: *androguard.decompiler.dad.instruction.RefExpression*

visit(*visitor*)

class androguard.decompiler.dad.instruction.**MonitorExitExpression**(*ref*)

Bases: *androguard.decompiler.dad.instruction.RefExpression*

visit(*visitor*)

class androguard.decompiler.dad.instruction.**MoveExceptionExpression**(*ref*, *_type*)

Bases: *androguard.decompiler.dad.instruction.RefExpression*

get_lhs()

get_used_vars()

has_side_effect()

replace_lhs(*new*)

visit(*visitor*)

class androguard.decompiler.dad.instruction.**MoveExpression**(*lhs*, *rhs*)

Bases: *androguard.decompiler.dad.instruction.IRForm*

get_lhs()

get_rhs()

get_used_vars()

has_side_effect()

is_call()

replace(*old*, *new*)

replace_lhs(*new*)

replace_var(*old*, *new*)

visit(*visitor*)

class androguard.decompiler.dad.instruction.**MoveResultExpression**(*lhs*, *rhs*)

Bases: *androguard.decompiler.dad.instruction.MoveExpression*

has_side_effect()

is_propagable()

visit(*visitor*)

class androguard.decompiler.dad.instruction.**NewArrayExpression**(*asize*, *atype*)

Bases: *androguard.decompiler.dad.instruction.ArrayExpression*

get_used_vars()

is_propagable()

replace(*old*, *new*)

replace_var(*old*, *new*)

visit(*visitor*)

class androguard.decompiler.dad.instruction.**NewInstance**(*ins_type*)

Bases: *androguard.decompiler.dad.instruction.IRForm*

get_type()

get_used_vars()

replace(*old*, *new*)

visit(*visitor*)

class androguard.decompiler.dad.instruction.**NopExpression**

Bases: *androguard.decompiler.dad.instruction.IRForm*

get_lhs()

get_used_vars()

visit(*visitor*)

class androguard.decompiler.dad.instruction.**Param**(*value*, *atype*)

Bases: *androguard.decompiler.dad.instruction.Variable*

is_const()

visit(*visitor*)

class androguard.decompiler.dad.instruction.**RefExpression**(*ref*)

Bases: *androguard.decompiler.dad.instruction.IRForm*

get_used_vars()

is_propagable()

replace(*old*, *new*)

replace_var(*old*, *new*)

class androguard.decompiler.dad.instruction.**ReturnInstruction**(*arg*)

Bases: *androguard.decompiler.dad.instruction.IRForm*

get_lhs()

get_used_vars()

replace(*old*, *new*)

replace_var(*old*, *new*)

visit(*visitor*)

class androguard.decompiler.dad.instruction.**StaticExpression**(*cls_name*, *field_type*, *field_name*)

Bases: *androguard.decompiler.dad.instruction.IRForm*

get_type()

replace(*old*, *new*)

visit(*visitor*)

class androguard.decompiler.dad.instruction.**StaticInstruction**(*rhs*, *klass*, *ftype*, *name*)

Bases: *androguard.decompiler.dad.instruction.IRForm*

get_lhs()

get_used_vars()

has_side_effect()

replace(*old*, *new*)

replace_var(*old*, *new*)

visit(*visitor*)

class androguard.decompiler.dad.instruction.**SwitchExpression**(*src*, *branch*)

Bases: *androguard.decompiler.dad.instruction.IRForm*

get_used_vars()

replace(*old*, *new*)

replace_var(*old*, *new*)

visit(*visitor*)

class androguard.decompiler.dad.instruction.**ThisParam**(*value*, *atype*)

Bases: *androguard.decompiler.dad.instruction.Param*

visit(*visitor*)

class androguard.decompiler.dad.instruction.**ThrowExpression**(*ref*)

Bases: *androguard.decompiler.dad.instruction.RefExpression*

visit(*visitor*)

class androguard.decompiler.dad.instruction.**UnaryExpression**(*op*, *arg*, *_type*)

Bases: *androguard.decompiler.dad.instruction.IRForm*

get_type()

get_used_vars()

replace(*old*, *new*)

replace_var(*old*, *new*)

visit(*visitor*)

class androguard.decompiler.dad.instruction.**Variable**(*value*)

Bases: *androguard.decompiler.dad.instruction.IRForm*

get_used_vars()

is_ident()

value()

visit(*visitor*)

visit_decl(*visitor*)

androguard.decompiler.dad.node module

class androguard.decompiler.dad.node.**Interval**(*head*)

Bases: object

add_node(*node*)

compute_end(*graph*)

get_end()

get_head()

class androguard.decompiler.dad.node.**LoopType**

Bases: object

copy()

property is_endless

property is_posttest

property is_pretest

class androguard.decompiler.dad.node.**MakeProperties**(*name*, *bases*, *dct*)

Bases: type

class androguard.decompiler.dad.node.**Node**(*name*)

Bases: object

copy_from(*node*)

get_end()

get_head()

update_attribute_with(*n_map*)

class androguard.decompiler.dad.node.**NodeType**

Bases: object

copy()

property is_cond

property is_return

property is_stmt

property is_switch

property is_throw

androguard.decompiler.dad.opcode_ins module

class androguard.decompiler.dad.opcode_ins.**Op**

Bases: object

ADD = '+'

AND = '&'

CMP = 'cmp'

DIV = '/'

EQUAL = '=='

GEQUAL = '>='

GREATER = '>'

INTSHL = '<<'

INTSHR = '>>'

LEQUAL = '<='

LONGSHL = '<<'

LONGSHR = '>>'

LOWER = '<'

MOD = '%'

MUL = '*'

NEG = '-'

NEQUAL = '!='

NOT = '~'

OR = '|'

SUB = '-'

XOR = '^'

androguard.decompiler.dad.opcode_ins.**adddouble**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**adddouble2addr**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**addfloat**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**addfloat2addr**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**addint**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**addint2addr**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**addintlit16**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**addintlit8**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**addlong**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**addlong2addr**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**aget**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**agetboolean**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**agetbyte**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**agetchar**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**agetobject**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**agetshort**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**agetwide**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**andint**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**andint2addr**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**andintlit16**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**andintlit8**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**andlong**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**andlong2addr**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**aput**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**aputboolean**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**aputbyte**(*ins*, *vmap*) androguard.decompiler.dad.opcode_ins.**aputchar**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**aputobject**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**aputshort**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**aputwide**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**arraylength**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**assign_binary_2addr_exp**(*ins*, *val_op*, *op_type*, *vmap*)

androguard.decompiler.dad.opcode_ins.**assign_binary_exp**(*ins*, *val_op*, *op_type*, *vmap*)

androguard.decompiler.dad.opcode_ins.**assign_cast_exp**(*val_a*, *val_b*, *val_op*, *op_type*, *vmap*)

androguard.decompiler.dad.opcode_ins.**assign_cmp**(*val_a*, *val_b*, *val_c*, *cmp_type*, *vmap*)

androguard.decompiler.dad.opcode_ins.**assign_const**(*dest_reg*, *cst*, *vmap*)

androguard.decompiler.dad.opcode_ins.**assign_lit**(*op_type*, *val_cst*, *val_a*, *val_b*, *vmap*)

androguard.decompiler.dad.opcode_ins.**checkcast**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**cmpgdouble**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**cmpgfloat**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**cmpldouble**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**cmplfloat**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**cmplong**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**const**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**const16**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**const4**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**constclass**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**consthigh16**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**conststring**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**conststringjumbo**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**constwide**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**constwide16**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**constwide32**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**constwidehigh16**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**divdouble**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**divdouble2addr**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**divfloat**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**divfloat2addr**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**divint**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**divint2addr**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**divintlit16**(*ins*, *vmap*) androguard.decompiler.dad.opcode_ins.**divintlit8**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**divlong**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**divlong2addr**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**doubletofloat**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**doubletoint**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**doubletolong**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**fillarraydata**(*ins*, *vmap*, *value*)

androguard.decompiler.dad.opcode_ins.**fillarraydatapayload**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**fillednewarray**(*ins*, *vmap*, *ret*)

androguard.decompiler.dad.opcode_ins.**fillednewarrayrange**(*ins*, *vmap*, *ret*)

androguard.decompiler.dad.opcode_ins.**floattodouble**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**floattoint**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**floattolong**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**get_args**(*vmap*, *param_type*, *largs*)

androguard.decompiler.dad.opcode_ins.**get_variables**(*vmap*, **variables*)

androguard.decompiler.dad.opcode_ins.**goto**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**goto16**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**goto32**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**ifeq**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**ifeqz**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**ifge**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**ifgez**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**ifgt**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**ifgtz**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**ifle**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**iflez**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**iflt**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**ifltz**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**ifne**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**ifnez**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**iget**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**igetboolean**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**igetbyte**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**igetchar**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**igetobject**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**igetshort**(*ins*, *vmap*) androguard.decompiler.dad.opcode_ins.**igetwide**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**instanceof**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**inttobyte**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**inttochar**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**inttodouble**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**inttofloat**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**inttolong**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**inttoshort**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**invokedirect**(*ins*, *vmap*, *ret*)

androguard.decompiler.dad.opcode_ins.**invokedirectrange**(*ins*, *vmap*, *ret*)

androguard.decompiler.dad.opcode_ins.**invokeinterface**(*ins*, *vmap*, *ret*)

androguard.decompiler.dad.opcode_ins.**invokeinterfacerange**(*ins*, *vmap*, *ret*)

androguard.decompiler.dad.opcode_ins.**invokestatic**(*ins*, *vmap*, *ret*)

androguard.decompiler.dad.opcode_ins.**invokestaticrange**(*ins*, *vmap*, *ret*)

androguard.decompiler.dad.opcode_ins.**invokesuper**(*ins*, *vmap*, *ret*)

androguard.decompiler.dad.opcode_ins.**invokesuperrange**(*ins*, *vmap*, *ret*)

androguard.decompiler.dad.opcode_ins.**invokevirtual**(*ins*, *vmap*, *ret*)

androguard.decompiler.dad.opcode_ins.**invokevirtualrange**(*ins*, *vmap*, *ret*)

androguard.decompiler.dad.opcode_ins.**iput**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**iputboolean**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**iputbyte**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**iputchar**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**iputobject**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**iputshort**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**iputwide**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**load_array_exp**(*val_a*, *val_b*, *val_c*, *ar_type*, *vmap*)

androguard.decompiler.dad.opcode_ins.**longtodouble**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**longtofloat**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**longtoint**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**monitorenter**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**monitorexit**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**move**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**move16**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**moveexception**(*ins*, *vmap*, *_type*)

androguard.decompiler.dad.opcode_ins.**movefrom16**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**moveobject**(*ins*, *vmap*) androguard.decompiler.dad.opcode_ins.**moveobject16**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**moveobjectfrom16**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**moveresult**(*ins*, *vmap*, *ret*)

androguard.decompiler.dad.opcode_ins.**moveresultobject**(*ins*, *vmap*, *ret*)

androguard.decompiler.dad.opcode_ins.**moveresultwide**(*ins*, *vmap*, *ret*)

androguard.decompiler.dad.opcode_ins.**movewide**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**movewide16**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**movewidefrom16**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**muldouble**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**muldouble2addr**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**mulfloat**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**mulfloat2addr**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**mulint**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**mulint2addr**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**mulintlit16**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**mulintlit8**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**mullong**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**mullong2addr**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**negdouble**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**negfloat**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**negint**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**neglong**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**newarray**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**newinstance**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**nop**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**notint**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**notlong**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**orint**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**orint2addr**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**orintlit16**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**orintlit8**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**orlong**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**orlong2addr**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**packedswitch**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**remdouble**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**remdouble2addr**(*ins*, *vmap*) androguard.decompiler.dad.opcode_ins.**remfloat**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**remfloat2addr**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**remint**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**remint2addr**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**remintlit16**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**remintlit8**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**remlong**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**remlong2addr**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**return_reg**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**returnobject**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**returnvoid**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**returnwide**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**rsubint**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**rsubintlit8**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**sget**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**sgetboolean**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**sgetbyte**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**sgetchar**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**sgetobject**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**sgetshort**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**sgetwide**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**shlint**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**shlint2addr**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**shlintlit8**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**shllong**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**shllong2addr**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**shrint**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**shrint2addr**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**shrintlit8**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**shrlong**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**shrlong2addr**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**sparseswitch**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**sput**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**sputboolean**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**sputbyte**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**sputchar**(*ins*, *vmap*) androguard.decompiler.dad.opcode_ins.**sputobject**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**sputshort**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**sputwide**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**store_array_inst**(*val_a*, *val_b*, *val_c*, *ar_type*, *vmap*)

androguard.decompiler.dad.opcode_ins.**subdouble**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**subdouble2addr**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**subfloat**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**subfloat2addr**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**subint**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**subint2addr**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**sublong**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**sublong2addr**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**throw**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**ushrint**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**ushrint2addr**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**ushrintlit8**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**ushrlong**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**ushrlong2addr**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**xorint**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**xorint2addr**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**xorintlit16**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**xorintlit8**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**xorlong**(*ins*, *vmap*)

androguard.decompiler.dad.opcode_ins.**xorlong2addr**(*ins*, *vmap*)

androguard.decompiler.dad.util module

androguard.decompiler.dad.util.**build_path**(*graph*, *node1*, *node2*, *path=None*)

Build the path from node1 to node2. The path is composed of all the nodes between node1 and node2, node1 excluded. Although if there is a loop starting from node1, it will be included in the path.

androguard.decompiler.dad.util.**common_dom**(*idom*, *cur*, *pred*)

androguard.decompiler.dad.util.**create_png**(*cls_name*, *meth_name*, *graph*, *dir_name='graphs2'*)

Creates a PNG from a given `Graph`.

Parameters

- **cls_name** (`str`) – name of the class
- **meth_name** (`str`) – name of the method
- **graph** (`androguard.decompiler.dad.graph.Graph`) –

- **dir_name** (*str*) – output directory

androguard.decompiler.dad.util.**get_access_class**(*access*)

androguard.decompiler.dad.util.**get_access_field**(*access*)

androguard.decompiler.dad.util.**get_access_method**(*access*)

androguard.decompiler.dad.util.**get_params_type**(*descriptor*)
Return the parameters type of a descriptor (e.g (IC)V)

androguard.decompiler.dad.util.**get_type**(*atype*, *size=None*)
Retrieve the java type of a descriptor (e.g : I)

androguard.decompiler.dad.util.**get_type_size**(*param*)
Return the number of register needed by the type @param

androguard.decompiler.dad.util.**merge_inner**(*clsdict*)
Merge the inner class(es) of a class: e.g class A { … } class A$foo{ … } class A$bar{ … } ==> class A { class foo{…} class bar{…} … }

androguard.decompiler.dad.writer module

class androguard.decompiler.dad.writer.**Writer**(*graph*, *method*)
Bases: object

Transforms a method into Java code.

dec_ind(*i=1*)

end_ins()

inc_ind(*i=1*)

space()

str_ext()

visit_alength(*array*)

visit_aload(*array*, *index*)

visit_assign(*lhs*, *rhs*)

visit_astore(*array*, *index*, *rhs*, *data=None*)

visit_base_class(*cls*, *data=None*)

visit_binary_expression(*op*, *arg1*, *arg2*)

visit_cast(*op*, *arg*)

visit_catch_node(*catch_node*)

visit_check_cast(*arg*, *atype*)

visit_cond_expression(*op*, *arg1*, *arg2*)

visit_cond_node(*cond*)

visit_condz_expression(*op*, *arg*)

visit_constant(*cst*)

visit_decl(*var*)

visit_fill_array(*array*, *value*)

visit_filled_new_array(*atype*, *size*, *args*)

visit_get_instance(*arg*, *name*, *data=None*)

visit_get_static(*cls*, *name*)

visit_ins(*ins*)

visit_invoke(*name*, *base*, *ptype*, *rtype*, *args*, *invokeInstr*)

visit_loop_node(*loop*)

visit_monitor_enter(*ref*)

visit_monitor_exit(*ref*)

visit_move(*lhs*, *rhs*)

visit_move_exception(*var*, *data=None*)

visit_move_result(*lhs*, *rhs*)

visit_new(*atype*, *data=None*)

visit_new_array(*atype*, *size*)

visit_node(*node*)

visit_nop()

visit_param(*param*, *data=None*)

visit_put_instance(*lhs*, *name*, *rhs*, *data=None*)

visit_put_static(*cls*, *name*, *rhs*)

visit_return(*arg*)

visit_return_node(*ret*)

visit_return_void()

visit_short_circuit_condition(*nnot*, *aand*, *cond1*, *cond2*)

visit_statement_node(*stmt*)

visit_super()

visit_switch(*arg*)

visit_switch_node(*switch*)

visit_this()

visit_throw(*ref*)

visit_throw_node(*throw*)

visit_try_node(*try_node*)

visit_unary_expression(*op*, *arg*)

visit_variable(*var*)

write(*s*, *data=None*)

write_ext(*t*)

write_ind()

write_ind_visit_end(*lhs*, *s*, *rhs=None*, *data=None*)

write_ind_visit_end_ext(*lhs*, *before*, *s*, *after*, *rhs=None*, *data=None*, *subsection='UNKNOWN_SUBSECTION'*)

write_inplace_if_possible(*lhs*, *rhs*)

write_method()

androguard.decompiler.dad.writer.**string**(*s*)

Convert a string to a escaped ASCII representation including quotation marks :param s: a string :return: ASCII escaped string

Module contents

Submodules

androguard.decompiler.decompiler module

class androguard.decompiler.decompiler.**DecompilerDAD**(*vm*, *vmx*)

Bases: object

display_all(*_class*)

display_source(*m*)

get_all(*class_name*)

get_ast_class(*_class*)

get_ast_method(*m*)

get_source_class(*_class*)

get_source_class_ext(*_class*)

get_source_method(*m*)

class androguard.decompiler.decompiler.**DecompilerDed**(*vm*, *bin_ded='ded.sh'*, *tmp_dir='/tmp/'*)

Bases: object

display_all(*_class*)

display_source(*method*)

get_all(*class_name*)

get_source_class(*_class*)

get_source_method(*method*)

class androguard.decompiler.decompiler.**DecompilerDex2Fernflower**(*vm*, *bin_dex2jar='dex2jar.sh'*, *bin_fernflower='fernflower.jar'*, *options_fernflower={'asc': '1', 'dgs': '1'}*, *tmp_dir='/tmp/'*)

Bases: object

display_all(*_class*)

display_source(*method*)

get_all(*class_name*)

get_source_class(*_class*)

get_source_method(*method*)

class androguard.decompiler.decompiler.**DecompilerDex2Jad**(*vm*, *bin_dex2jar='dex2jar.sh'*, *bin_jad='jad'*, *tmp_dir='/tmp/'*)

Bases: object

display_all(*_class*)

display_source(*method*)

get_all(*class_name*)

get_source_class(*_class*)

get_source_method(*method*)

class androguard.decompiler.decompiler.**DecompilerDex2WineJad**(*vm*, *bin_dex2jar='dex2jar.sh'*, *bin_jad='jad'*, *tmp_dir='/tmp/'*)

Bases: object

display_all(*_class*)

display_source(*method*)

get_all(*class_name*)

get_source_class(*_class*)

get_source_method(*method*)

class androguard.decompiler.decompiler.**DecompilerJADX**(*vm*, *vmx*, *jadx='jadx'*, *keepfiles=False*)

Bases: object

display_all(*_class*)

???

Parameters **_class** –

Returns

display_source(*m*)

This method does the same as *get_source_method* but prints the result directly to stdout

Parameters **m** – *EncodedMethod* to print

Returns

get_all(*class_name*)

???

Parameters **class_name** –

Returns

get_source_class(*_class*)

Return the Java source code of a whole class

Parameters **_class** – *ClassDefItem* object, to get the source from

Returns

get_source_method(*m*)
Return the Java source of a single method

Parameters **m** – *EncodedMethod* Object

Returns

class androguard.decompiler.decompiler.**Dex2Jar**(*vm*, *bin_dex2jar='dex2jar.sh'*, *tmp_dir='/tmp/'*)
Bases: object

get_jar()

exception androguard.decompiler.decompiler.**JADXDecompilerError**
Bases: Exception

Exception for JADX related problems

class androguard.decompiler.decompiler.**MethodFilter**(***options*)
Bases: pygments.filter.Filter

filter(*lexer*, *stream*)

Module contents

3.1.2 Submodules

3.1.3 androguard.misc module

androguard.misc.**AnalyzeAPK**(*_file*, *session=None*, *raw=False*)
Analyze an android application and setup all stuff for a more quickly analysis! If session is None, no session is used at all. This is the default behaviour. If you like to continue your work later, it might be a good idea to use a session. A default session can be created by using *get_default_session()*.

Parameters

- **_file** (*string (for filename) or bytes (for raw)*) – the filename of the android application or a buffer which represents the application
- **session** – A session (default: None)
- **raw** – boolean if raw bytes are supplied instead of a filename

Return type return the *APK*, list of *DalvikVMFormat*, and *Analysis* objects

androguard.misc.**AnalyzeDex**(*filename*, *session=None*)
Analyze an android dex file and setup all stuff for a more quickly analysis !

Parameters

- **filename** (*string*) – the filename of the android dex file or a buffer which represents the dex file
- **session** – A session (Default None)

Return type return a tuple of (sha256hash, DalvikVMFormat, Analysis)

androguard.misc.**AnalyzeODex**(*filename*, *session=None*)
Analyze an android odex file and setup all stuff for a more quickly analysis !

Parameters

- **filename** (*string*) – the filename of the android dex file or a buffer which represents the dex file
- **session** – The Androguard Session to add the ODex to (default: None)

Return type return a tuple of (sha256hash, DalvikOdexVMFormat, Analysis)

androguard.misc.**RunDecompiler**(*d*, *dx*, *decompiler_name*)

Run the decompiler on a specific analysis

Parameters

- **d** (DalvikVMFormat object) – the DalvikVMFormat object
- **dx** (VMAnalysis object) – the analysis of the format
- **decompiler** (*string*) – the type of decompiler to use ("dad", "dex2jad", "ded")

androguard.misc.**clean_file_name**(*filename*, *unique=True*, *replace='_'*, *force_nt=False*)

Return a filename version, which has no characters in it which are forbidden. On Windows these are for example <, /, ?, …

The intention of this function is to allow distribution of files to different OSes.

Parameters

- **filename** – string to clean
- **unique** – check if the filename is already taken and append an integer to be unique (default: True)
- **replace** – replacement character. (default: '_')
- **force_nt** – Force shortening of paths like on NT systems (default: False)

Returns clean string

androguard.misc.**get_default_session**()

Return the default Session from the configuration or create a new one, if the session in the configuration is None.

Return type *androguard.session.Session*

androguard.misc.**init_print_colors**()

androguard.misc.**sign_apk**(*filename*, *keystore*, *storepass*)

Use jarsigner to sign an APK file.

Parameters

- **filename** – APK file on disk to sign (path)
- **keystore** – path to keystore
- **storepass** – your keystorage passphrase

3.1.4 androguard.session module

androguard.session.**Load**(*filename*)

load your session!

example:

```
s = session.Load("mysession.ag")
```

Parameters **filename** (*string*) – the filename where the session has been saved

Return type the elements of your session :)

androguard.session.**Save**(*session*, *filename=None*)

save your session to use it later.

Returns the filename of the written file. If not filename is given, a file named *androguard_session_<DATE>.ag* will be created in the current working directory. *<DATE>* is a timestamp with the following format: *%Y-%m-%d_%H%M%S*.

This function will overwrite existing files without asking.

If the file could not written, None is returned.

example:

```
s = session.Session()
session.Save(s, "msession.ag")
```

Parameters

- **session** – A Session object to save
- **filename** (*string*) – output filename to save the session

class androguard.session.**Session**(*export_ipython=False*)

Bases: object

A Session is able to store multiple APK, DEX or ODEX files and can be pickled to disk in order to resume work later.

The main function used in Sessions is probably *add()*, which adds files to the session and performs analysis on them.

Afterwards, the files can be gathered using methods such as *get_objects_apk()*, *get_objects_dex()* or *get_classes()*.

example:

```
s = Session()
digest = s.add("some.apk")

print("SHA256 of the file: {}".format(digest))

a, d, dx = s.get_objects_apk("some.apk", digest)
print(a.get_package())

# Reset the Session for a fresh set of files
s.reset()

digest2 = s.add("classes.dex")
print("SHA256 of the file: {}".format(digest2))
for h, d, dx in s.get_objects_dex():
    print("SHA256 of the DEX file: {}".format(h))
```

add(*filename*, *raw_data=None*, *dx=None*)

Generic method to add a file to the session.

This is the main method to use when adding files to a Session!

If an APK file is supplied, all DEX files are analyzed too. For DEX and ODEX files, only this file is analyzed (what else should be analyzed).

Returns the SHA256 of the analyzed file.

Parameters

- **filename** – filename to load
- **raw_data** – bytes of the file, or None to load the file from filename
- **dx** – An already exiting `Analysis` object

Returns the sha256 of the file or None on failure

addAPK (*filename*, *data*)

Add an APK file to the Session and run analysis on it.

Parameters

- **filename** – (file)name of APK file
- **data** – binary data of the APK file

Returns a tuple of SHA256 Checksum and APK Object

addDEX (*filename*, *data*, *dx=None*, *postpone_xref=False*)

Add a DEX file to the Session and run analysis.

Parameters

- **filename** – the (file)name of the DEX file
- **data** – binary data of the dex file
- **dx** – an existing Analysis Object (optional)
- **postpone_xref** – True if no xref shall be created, and will be called manually

Returns A tuple of SHA256 Hash, DalvikVMFormat Object and Analysis object

addDEY (*filename*, *data*, *dx=None*)

Add an ODEX file to the session and run the analysis

get_all_apks ()

Yields a list of tuples of SHA256 hash of the APK and APK objects of all analyzed APKs in the Session.

get_analysis (*current_class*)

Returns the `Analysis` object which contains the *current_class*.

Parameters **current_class** (`androguard.core.bytecodes.dvm.ClassDefItem`) – The class to search for

Return type *androguard.core.analysis.analysis.Analysis*

get_classes ()

Returns all Java Classes from the DEX objects as an array of DEX files.

get_digest_by_class (*current_class*)

Return the SHA256 hash of the object containing the ClassDefItem

Returns the first digest this class was present. For example, if you analyzed an APK, this should return the digest of the APK and not of the DEX file.

get_filename_by_class (*current_class*)

Returns the filename of the DEX file where the class is in.

Returns the first filename this class was present. For example, if you analyzed an APK, this should return the filename of the APK and not of the DEX file.

Parameters **current_class** – ClassDefItem

Returns None if class was not found or the filename

get_format (*current_class*)

Returns the `DalvikVMFormat` of a given `ClassDefItem`.

Parameters **current_class** – A ClassDefItem

get_nb_strings ()

Return the total number of strings in all Analysis objects

get_objects_apk (*filename=None*, *digest=None*)

Returns APK, DalvikVMFormat and Analysis of a specified APK.

You must specify either *filename* or *digest*. It is possible to use both, but in this case only *digest* is used.

example:

```
s = Session()
digest = s.add("some.apk")
a, d, dx = s.get_objects_apk(digest=digest)
```

example:

```
s = Session()
filename = "some.apk"
digest = s.add(filename)
a, d, dx = s.get_objects_apk(filename=filename)
```

Parameters

- **filename** – the filename of the APK file, only used of digest is None
- **digest** – the sha256 hash, as returned by `add()` for the APK

Returns a tuple of (APK, [DalvikVMFormat], Analysis)

get_objects_dex ()

Yields all dex objects inclduing their Analysis objects

Returns tuple of (sha256, DalvikVMFormat, Analysis)

get_strings ()

Yields all StringAnalysis for all unique Analysis objects

isOpen ()

Test if any file was analyzed in this session

Returns *True* if any file was analyzed, *False* otherwise

reset ()

Reset the current session, delete all added files.

save (*filename=None*)

Save the current session, see also `Save()`.

show ()

Print information to stdout about the current session. Gets all APKs, all DEX files and all Analysis objects.

3.1.5 androguard.util module

androguard.util.**get_certificate_name_string** (*name*, *short=False*, *delimiter=’, ’*)

Format the Name type of a X509 Certificate in a human readable form.

Parameters

- **name** (dict or `asn1crypto.x509.Name`) – Name object to return the DN from
- **short** (*boolean*) – Use short form (default: False)
- **delimiter** (*str*) – Delimiter string or character between two parts (default: ', ')

Return type str

`androguard.util.`**`read`**(*filename*, *binary=True*)

Open and read a file

Parameters

- **filename** – filename to open and read
- **binary** – True if the file should be read as binary

Returns bytes if binary is True, str otherwise

3.1.6 Module contents

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX