ANDROID APPLICATION PACKAGE

Bitdefender

Android application package (APK)

- Apps mainly written using Java, Kotlin, C++ and recently Dart.
- code, data and resource files that comprise your app are compiled into an APK, an *Android package*, which is a ZIP archive file with an .apk suffix.

- APK structure
  - binary data and resources are in several directories and standard files within the zip container
    - e.g. AndroidManifest.xml, classes.dex
  - logic is constituted of app components
    - essential building blocks of an app.
    - components are entry points through which the system or a user can enter your app.

Application file structure

- **AndroidManifest.xml**
  - essential information about your app, *(e.g. package name, version, permissions and components)*
- **classes.dex**
  - compiled classes code in the DEX file format. Can be more than one.
- **META-INF/**
  - files required for validation with V1 signature schema
- **lib/**
  - compiled native code libraries specific to each CPU architecture *(e.g. armeabi, x86 or x86_64)*
- resources.arsc
  - compiled resources *(e.g. content from several files and XMLs along with language strings and styles)*
- res/
  - resources that aren't compiled into resources.arsc *(e.g. images)*
- assets/
  - app's assets *(malware often stores configuration files here)*

Manifest data file - **AndroidManifest.xml**

- Mandatory exist in the APK’s root directory
- is in the Android's binary XML.
  - Must use a decoder in order to analyze (e.g. apktool)
- manifest declares several key elements of an APK. Most relevant
  - The application package name
  - Permissions the application requires
  - Minimum (and targeted) API Level required by the app in order to install
  - The components of the application

Manifest data file - **key elements**

- **Package name**
  - Unique string app identifier
  - Found in the **<manifest>** XML element, in the **package** attribute

```
1  <?xml version="1.0" encoding="utf-8" standalone="no"?><manifest xmlns:android="http://schemas.android.com/apk/res/android" android:
   compileSdkVersion="30" android:compileSdkVersionCodename="11" package="com.facebook.lite" platformBuildVersionCode="30"
   platformBuildVersionName="11">
2      <supports-screens android:anyDensity="true" android:largeScreens="true" android:normalScreens="true" android:smallScreens="true"/>
3      <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
4      <uses-permission android:name="android.permission.INTERNET"/>
5      <uses-permission android:name="android.permission.WAKE_LOCK"/>
```

- **Minimum and targeted API level**
- in **<uses-sdk>** element at the attributes:
  - **android:minSdkVersion**: minimum level API at which the application installs
  - **android:targetSdkVersion**: intended API level for best performance of the app

```
platformBuildVersionName="11">
<uses-sdk
    android:minSdkVersion="26"
    android:targetSdkVersion="30" />
<supports-screens
```

Manifest data file - **key elements**

- **Permissions**
  - all permissions are, in one form or another required in the manifest.
  - install time and runtime granted permissions are declared using the **<uses-permission>** or **<uses-permission-sdk-23>** XML tags.
    - Runtime permissions also require extra code to be executed in order to be requested
  - special permissions are partially declared in the AndroidManifest.xml but it differs from case to case.

```
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE"/>
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
<uses-permission android:name="android.permission.GET_TASKS"/>
<uses-permission android:name="android.permission.READ_CALENDAR"/>
<uses-permission-sdk-23 android:name="android.permission.MODIFY_AUDIO_SETTINGS"/>
```

Components and entry points

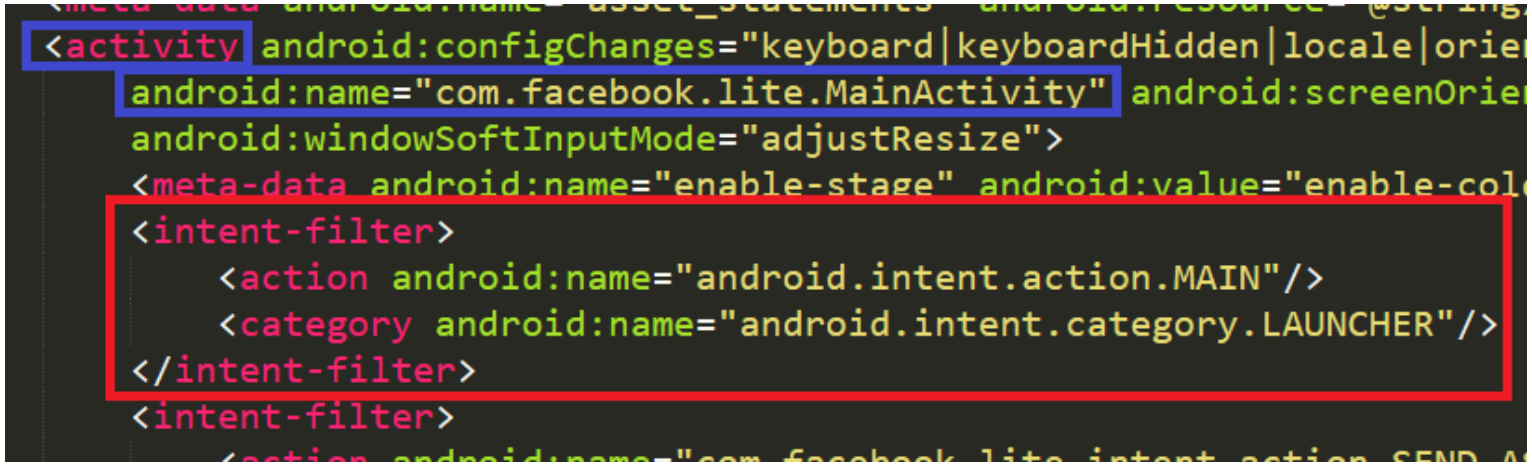
- Components are
  - essential building blocks of an app; code that is executed
  - entry points through which the system or a user can enter your app
  - each required component (that exists in code) must have an entry in the AndroidManifest.xml in order to be recognized by the OS

- Component types
  - Activities
  - Services
  - Broadcast receivers
  - Content providers
  - Application (often overlooked entry point)

Components - Activities

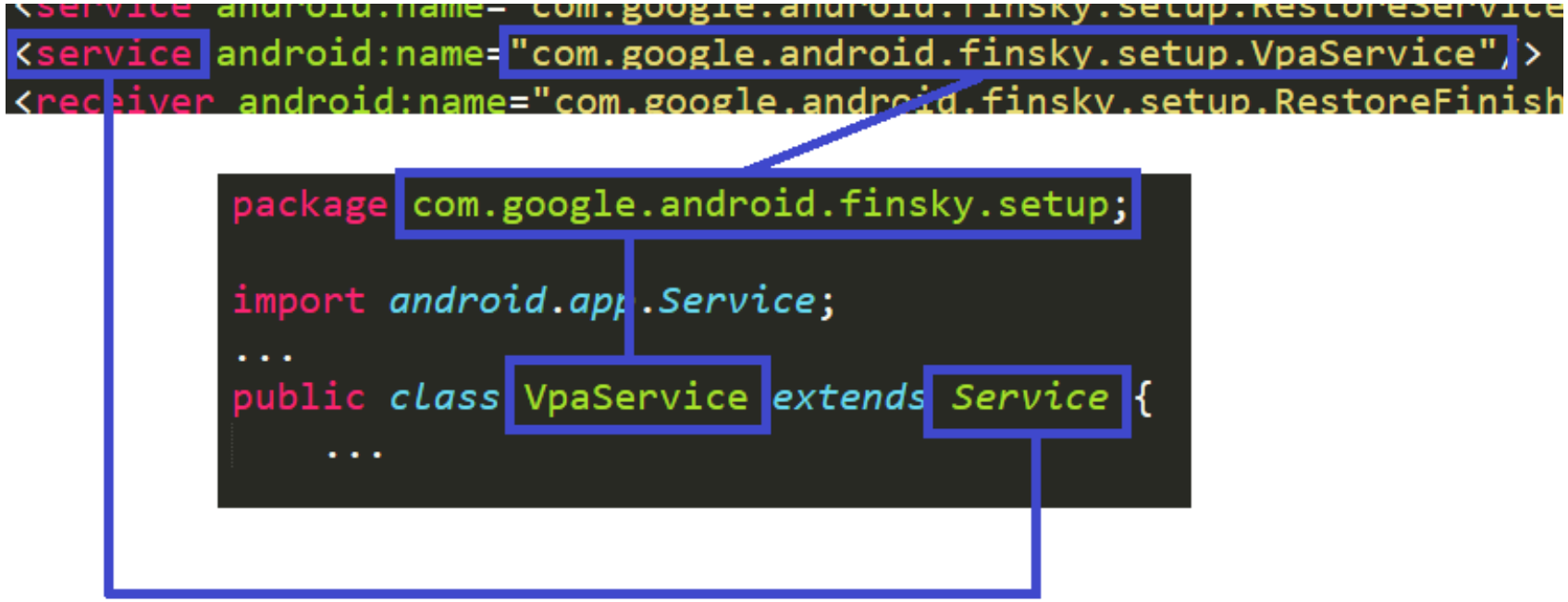
- Description: entry point for interacting with the user. It represents a single screen with a user interface.
- code: subclass of the **Activity** class
- manifest: **<activity>** element
- When tapping the icon of an application on the launcher, a specially designated ***Launcher Activity*** will be executed.

- **Launcher Activity** requires intent filter for
  - action MAIN
  - category LAUNCHER



Components - Services

- Description: general-purpose entry point that runs in the background to perform long-running operations. Does not provide a user interface. Some types of service require a notification to be shown indicating that the service is running
- Code: subclass of **Service** class
- Manifest: **<service>** element



Components – Broadcast receivers

- Description: enables the system to deliver *events* to the app outside of a regular user flow, allowing the app to respond to system-wide broadcast announcements. No UI.
- Code: subclass of **BroadcastReceiver** class
- Manifest: **<receiver>**
- Receivers specify which intents (events) they are listening to via the **<action>** element in the **<intent-filter>**. Intents can be system defined or user defined (custom)

```
<receiver android:enabled="false" android:exported="true" android:name="com.facebook.rti.push.service.MqttSystemBroadcastReceiver"
android:process=":fbns">
    <intent-filter>
        <action android:name="android.intent.action.MY_PACKAGE_REPLACED"/>
        <action android:name="android.intent.action.BOOT_COMPLETED"/>
        <action android:name="android.net.conn.CONNECTIVITY_CHANGE"/>
        <action android:name="android.intent.action.USER_PRESENT"/>
    </intent-filter>
</receiver>
```

Components – Content providers

- Description: gives structured access to data managed by the application to other applications; data sharing mechanism
- Code: subclass of **ContentProvider** class
- Manifest: **<provider>**
- Usually not used by malware ITW as malware does not need to provide a data sharing mechanism

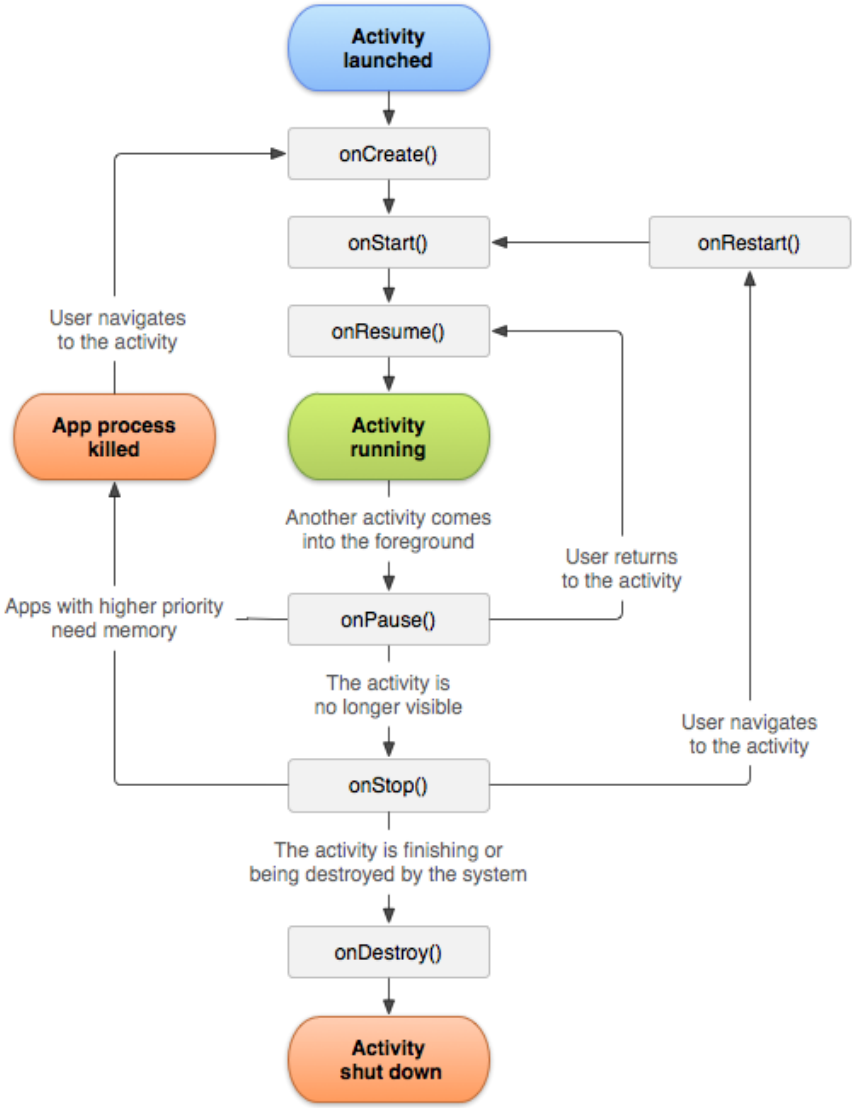
Application class entry point

- Description
  - a very important entry point and component but often overlooked
  - only one or none can exist for any given application
  - when the application is started it is started before any other component
  - Configurations are usually set here, and packers often use this entry point to decrypt the other components/payload/code.
- Code: subclass of **Application** class
- Manifest: in the **<application>** element, in the **name** attribute

```
<application android:allowBackup="false" android:appComponentFactory="androidx.core.app.CoreComponentFactory" android:de
android:hardwareAccelerated="true" android:icon="@mipmap/ic_launcher" android:label="@string/app_short_name" android:man
com.facebook.lite.storagemanager.ManageStorageActivity" android:name="com.facebook.lite.ClientApplicationSplittedShell"
networkSecurityConfig="@xml/fb_network_security_config" android:requestLegacyExternalStorage="true" android:supportsRtl=
="@style/whiteBackgroundTheme" android:vmSafeMode="@bool/disableJit">
    <meta-data android:name="com.facebook.build_rule" android:value="android_fblite_xzs_x86_release_fbsign"/>
```

Components – lifecycle and callbacks

- Each component has a specific set of methods that are called when the component is engaged.
- Which methods, their order and when they are executed compose their lifecycle
- Important general component callbacks
  - **onCreate** – specific to *all components*
    - first method called automatically (after class constructor)
  - **onReceive** – specific to *Broadcast Receivers*
    - Called when an event was received
  - **onStartCommand** – specific to *Services*
    - called when explicitly started by another component



Execution mechanisms

- Code can be executed in several ways
  - Dalvik Executable file (.dex)
  - Native code (usually written in C/C++)
  - Any external 3rd party language/execution mechanism if ported and run-on Android
    - Some popular examples: JavaScript, C#, Lua

Execution mechanisms - Dalvik Executable file (.dex)

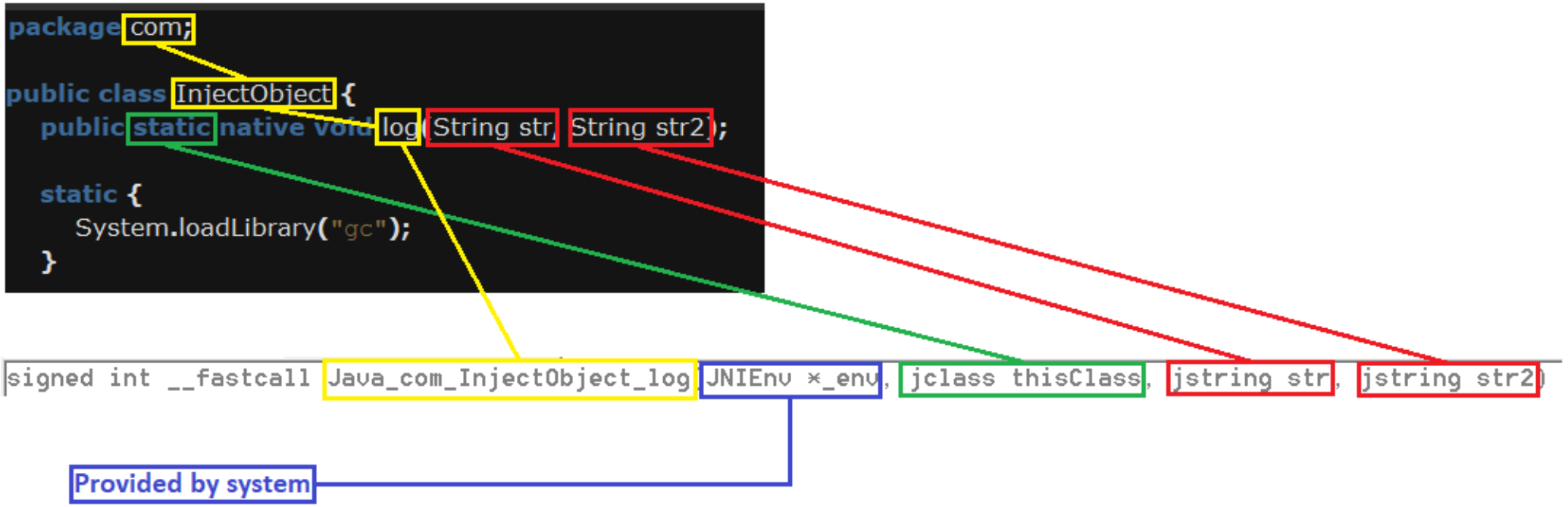
- Main code execution mechanism in APKs is via Dalvik Executable file (.dex)
  - Component code is located here
  - Custom format, similar to Java .class file format
  - Compiled in a file called **classes.dex**
  - More than 1 classes.dex can exist (classes2.dex, …, classesN.dex)
  - **It is possible to load more DEX files dynamically via API calls**
  - *By analyzing the classes.dex files it is possible to determine the compiler (tool) used to generate that file.*

Execution mechanisms - **Native code**

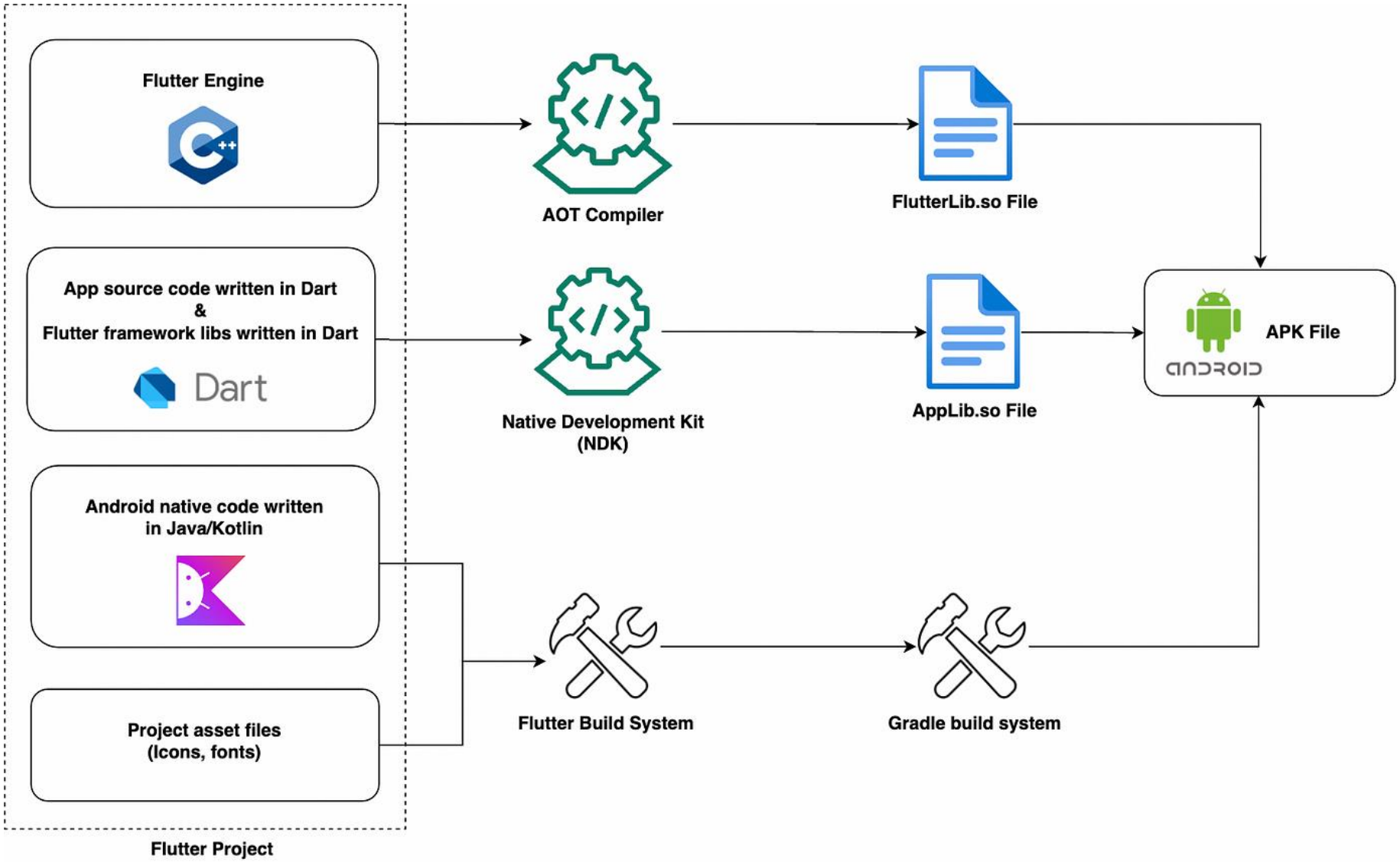
- code can be written in C/C++ and compiled into native libraries (.so files)
- executed via Java Native Interface (JNI). In rare cases, the entire app can be pure native
- compiled native code libraries specific to each CPU architecture (e.g. armeabi, x86 or x86_64) must exist for each targeted platform.
- Compiled native code libraries are found in the **lib** APK directory

Execution mechanisms - **Native code**

- Native methods are identified in Java by the keyword native
- In native, entry points method from Java layer are identified by their name ***Java_<package_name_and_class_name>***
- Argument mapping from Java to JNI follows the standard JNI convention, example:



Execution mechanisms - Native Code - Flutter



Execution mechanisms - JavaScript

- JavaScript can be executed from the Java layer via **loadUrl** API method from the **WebView** class

```
webView.loadUrl("javascript:(function f() { } )()")
```

- There are several libraries that take advantage of this possibility and extend the functionality by loading direct .html files with JavaScript in them
  - A few popular examples: Cordova, React Native, Mobile Angular UI, Android JS
- JavaScript/.html files are usually found in the /assets folder

CERT data file

- files required for validation with V1 signature schema (JAR signing) are found in **META-INF/**
  - **Signature can be verified using**
    - jarsigner -verify -verbose -certs <sample.apk> (only for V1)
    - apksigner verify -verbose --print-certs <sample.apk> (regardless of signature schema)
  - **Information about certificate can be extracted (only for V1)**
    - keytool -printcert -jarfile <sample.apk>
    - openssl pkcs7 -inform DER -in <V1CERTFILE.RSA/DSA/ECL> -noout -print_certs -text