Debugging Dalvik programs with IDA

Preface

Starting with v6.6 IDA Pro can debug Android applications written for the Dalvik Virtual Machine. It includes the source level debugging too. This tutorial explains how to set up and run a Dalvik debugging session.

Install Android SDK

First of all we have to install the Android SDK from the official site http://developer.android.com/sdk.

We do not need the ADT Bundle, so selecting the “SDK Tools Only” download is enough.

Environment variables

IDA needs to know where the *adb* utility resides, and tries various methods to locate it automatically. Usually IDA finds the path to *adb,* but if it fails then we can define the *ANDROID_SDK_HOME* or the *ANDROID_HOME* environment variable to point to the directory where the Android SDK is installed to.

Android device

Start the Android Emulator or connect the Android device to a USB port.

The information about creating AVDs (Android Virtual Devices) and starting the emulator can be found at the official site: Using the Emulator.

The information about preparing a physical device for development can be found at Using Hardware Devices.

Check that the device can be correctly detected by *adb*:

```
$ adb devices
List of devices attached
emulator-5554   device
```

Install application

IDA presumes that the debugged application is already installed on the Android emulator/device.

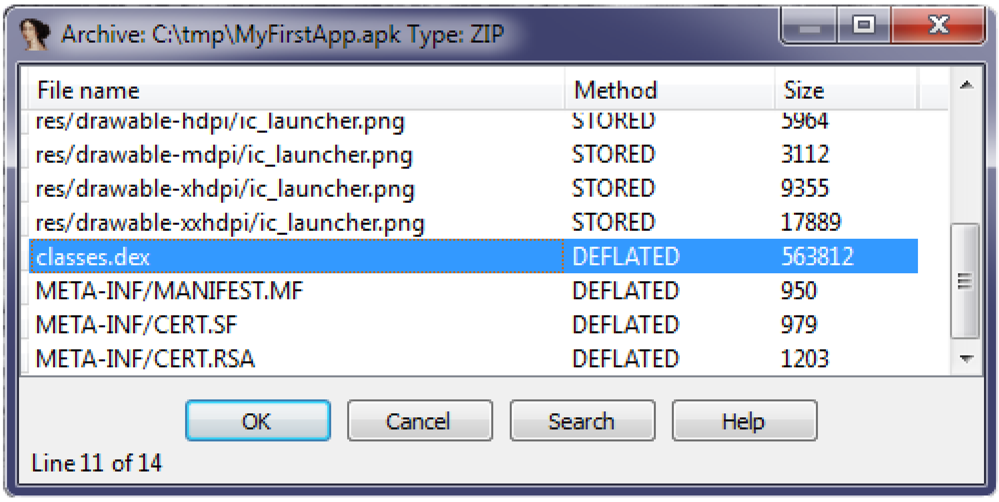
Please download *MyFirstApp.apk* and *MyFirstApp.src.zip* from our site. We will use this application in the tutorial.

We will use *adb* to install the application:

```
adb –s emulator-5554 install MyFirstApp.apk
```

Loading application into IDA

We can start with *apk* or *dex* files into IDA. If we specify the *apk* file, IDA will display its contents and ask us to select the desired file from the package. We select the *classes.dex* file:



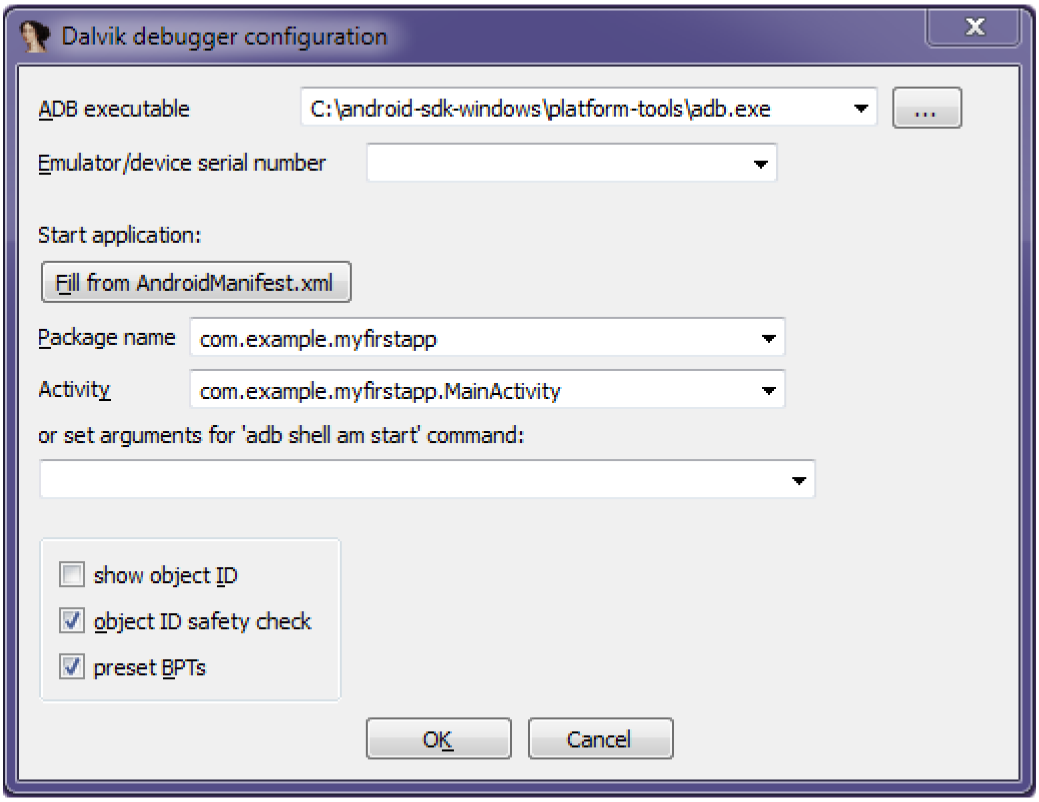
Archive: C:\tmp\MyFirstApp.apk Type: ZIP

File name	Method	Size
res/drawable-hdpi/ic_launcher.png	STORED	5964
res/drawable-mdpi/ic_launcher.png	STORED	3112
res/drawable-xhdpi/ic_launcher.png	STORED	9355
res/drawable-xxhdpi/ic_launcher.png	STORED	17889
classes.dex	DEFLATED	563812
META-INF/MANIFEST.MF	DEFLATED	950
META-INF/CERT.SF	DEFLATED	979
META-INF/CERT.RSA	DEFLATED	1203

Line 11 of 14

Dalvik debugger options

Before launching the debugger let us check out the debugger configuration. Go to "*Debugger/Debugger Options/Set specific options*":



Dalvik debugger configuration

ADB executable: C:\android-sdk-windows\platform-tools\adb.exe

Emulator/device serial number:

Start application:

Package name: com.example.myfirstapp

Activity: com.example.myfirstapp.MainActivity

or set arguments for 'adb shell am start' command:

□ show object ID
☑ object ID safety check
☑ preset BPTs

ADB executable

As mentioned above IDA tries to locate the *adb* utility. If IDA failed to find it then we can set the path to *adb* here.

Fill from AndroidManifest.xml

Press button and point IDA to *MyFirstApp.apk* file. IDA fetches package name and application activity name to start with.

Preset BPTs

IDA can help us to start debugging by setting breakpoints for all methods of the application activity class.

Path to sources

To use source-level debugging we have to set paths to the application source files. We can do it using the “*Options/Sources path …*” menu item.

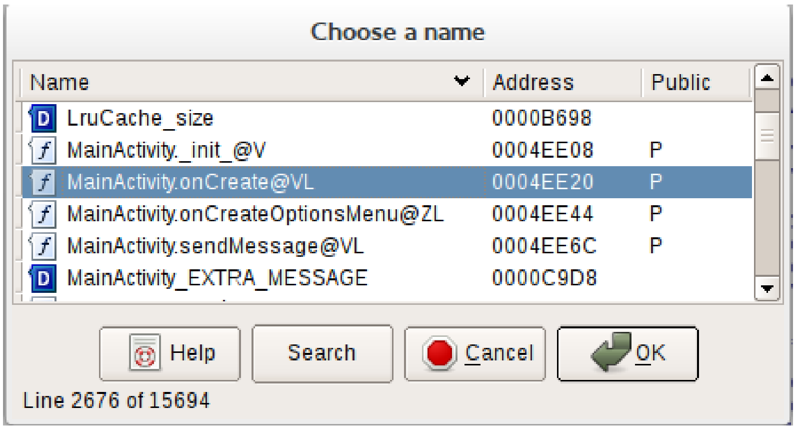
Our Dalvik debugger presumes that the application sources reside in the current (“.”) directory. If this is not the case, we can map current directory (“.”) to the directory where the source files are located.

Let us place the source files *DisplayMessageActivity.java* and *MainActivity.java* in the same directory as the *MyFirstApp.apk* package. This way we do not need any mapping.

Set breakpoints

Before launching the application it is reasonable to set a few breakpoints. We can rely on the decision made by IDA (see above the *presetBPTs* option) or set breakpoints ourselves. A good candidate is the “*onCreate*” method of the application's main activity.

We can use the activity name and the method name “*onCreate*” to set a breakpoint:



Choose a name

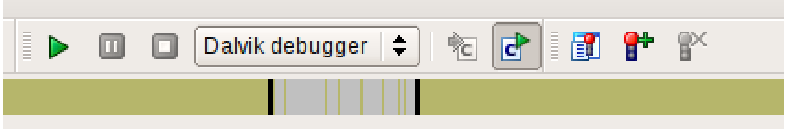
Name	Address	Public
LruCache_size	0000B698	
MainActivity._init_@V	0004EE08	P
MainActivity.onCreate@VL	0004EE20	P
MainActivity.onCreateOptionsMenu@ZL	0004EE44	P
MainActivity.sendMessage@VL	0004EE6C	P
MainActivity_EXTRA_MESSAGE	0000C9D8	

Line 2676 of 15694

Naturally, we can set any other breakpoints any time. For example, we can do it later, when we suspend the application.

Starting the debugger

At last we can start the debugger. Check that the Dalvik debugger backend is selected. Usually it should be done automatically by IDA:



If the debugger backend is correct, we are ready to start a debugger session. There are two ways to do it:

1. Launch a new copy of the application (Start process)
2. Attach to a running process (Attach to process)

1. Start process

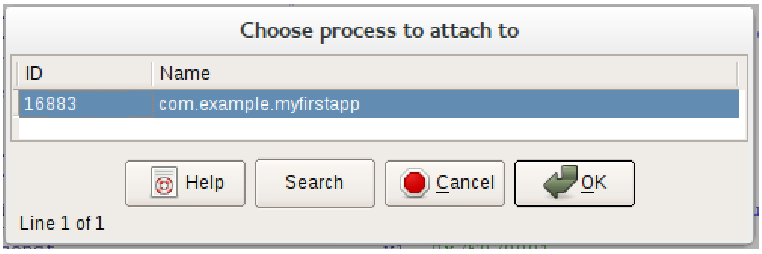
To start a new copy of the application just press <F9> or use the "*Debugger/Start process*" menu item. The Dalvik debugger will launch the application, wait until application is ready and open a debugger session to it.

We may wait for the execution to reach a breakpoint or press the "*Cancel*" button to suspend the application.

In our case let us wait until execution reach of *onCreate()* method breakpoint.

2. Attach to process

Instead of launching a new process we could attach to a running process and debug it. For that we could have selected the "*Debugger/Attach to process…*" menu item. IDA will display a list of active processes.


Choose process to attach to

ID	Name
16883	com.example.myfirstapp

Line 1 of 1

We just select the process we want to attach to.

Particularities of Dalvik debugger

All traditional debug actions like Step into, Step over, Run until return and others can be used. If the application sources are accessible then IDA will automatically switch to the source-level debugging.

Below is the list of special things about our Dalvik debugger:

- In Dalvik there is no stack and there is no *SP* register. The only available register is *IP*.
- The method frame registers and slots (*v0*, *v1*, …) are represented as local variables in IDA. We can see them in the *Debugger/Debugger Windows/Locals* window (see below)

- The stack trace is available from "*Debugger/Windows/Stack trace*" (the hot key is <Ctrl-Alt-S>).

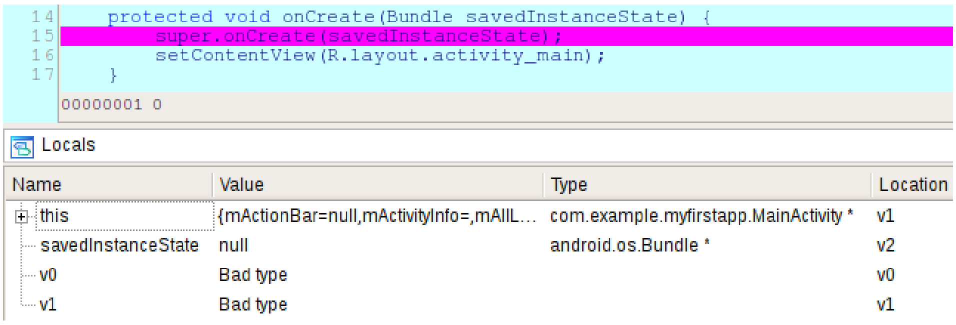
- When the application is running, it may execute some system code. If we break the execution by clicking on the "*Cancel*" button, quite often we may find ourselves outside of the application, in the system code. The value of the *IP* register is *0xFFFFFFFF* in this case, and stack trace shows only system calls and a lot of *0xFFFFFFFF*s. It means that IDA could not locate the current execution position inside the application. We recommend to set more breakpoints inside the application, resume the execution and interact with application by clicking on its windows, selecting menu items, etc. The same thing can occur when we step out the application.

- Use "Run until return" command to return to the source-level debugging if you occasionally step into a method and the value of the *IP* register becomes *0xFFFFFFFF*.

Locals window

IDA considers the method frame registers, slots, and variables (*v0*, *v1*, …) as local variables. To see their values we have to open the "*Locals*" window from the "*Debugger/Debugger windows/Locals*" menu item.

At the moment the debugger stopped the execution at the breakpoint which we set on *onCreate()* method. Let us open the "*Locals*" window and we will see something like the following:



```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
00000001 0
```

Locals

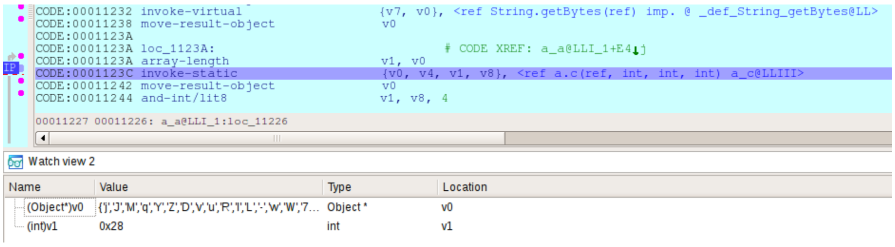
Name	Value	Type	Location
⊞ this	{mActionBar=null,mActivityInfo=,mAllL…	com.example.myfirstapp.MainActivity *	v1
savedInstanceState	null	android.os.Bundle *	v2
v0	Bad type		v0
v1	Bad type		v1

If the information about the frame is available (the symbol table is intact) then IDA shows the method arguments, the method local variables with names and other non-named variables. Otherwise some variable values will not be displayed because IDA does not know their types.

Variables without type information are marked with "*Bad type*" in the "*Locals*" window. To see the variable value in this case please use the "*Watch view*" window (see below).

Watch view window

To open the "*Watch view*" window please select the "*Debugger/Debugger windows/Watch view*" menu item. In this window we can add any variable to watch its value.



```
CODE:00011232 invoke-virtual              {v7, v0}, <ref String.getBytes(ref) imp. @ _def_String_getBytes@LL>
CODE:00011238 move-result-object          v0
CODE:0001123A
CODE:0001123A loc_1123A:                          # CODE XREF: a_a@LLI_1+E4↓j
CODE:0001123A array-length                v1, v0
CODE:0001123C invoke-static               {v0, v4, v1, v8}, <ref a.c(ref, int, int, int) a_c@LLIII>
CODE:00011242 move-result-object          v0
CODE:00011244 and-int/lit8                v1, v8, 4

00011227 00011226: a_a@LLI_1:loc_11226
```

Watch view 2

Name	Value	Type	Location
(Object*)v0	{'j','J','M','q','Y','Z','D','V','u','R','I','L','-','w','W','7...	Object *	v0
(int)v1	0x28	int	v1

Please note that we have to specify type of variable if it is not known. Use C-style casts:

```
(Object*)v0
(String)v6
(char*)v17
(int)v7
```

We do not need to specify the real type of an object variable, the "*(Object*)*" cast is enough. IDA can derive the real object type itself.

Attention! An incorrect type may cause the Dalvik VM to crash. There is not much we can do about it. Our recommendation is to never cast an integer variable to an object type, the Dalvik VM usually crashes if we do that. But the integer cast "*(int)*" is safe in practice. If the "object ID safety check" debugger option is selected then IDA tries to prevent such a situation. The only drawback is that this option adds some overhead.

Keeping the above in the mind, do not leave the cast entries in the "*Watch view*" window for a long time. Delete them before any executing instruction that may change the type of the watched variable.

If something goes wrong

- Check the path to *adb* in the "*Debugger specific options*"
- Check the package and activity names
- Check that the emulator is working and was registered as an *adb* device. Try to restart the *adb* daemon.
- Check that the application was successfully installed on the emulator/device
- Check the output window of IDA for any errors or warnings
- Turn on more debug print in IDA with the *-z50000* command line switch.