Debugging Dalvik Programs

Table of Contents

1. Preface
2. Installing Android Studio
3. Environment Variables 1
4. Android Device
5. Installing the App
6. Loading Application into IDA
7. Dalvik Debugger Options 2
7.1. Connection Settings 3
7.2. Start Application 3
7.3. Detect Local Variable Types 4
7.4. Other Options
8. Path to Sources 5
9. Setting Breakpoints
10. Starting the Debugger
10.1. Launching the App
10.2. Attaching to a Running App
11. Particularities of Dalvik Debugger
11.1. Locals Window
11.2. Watch View Window 9
12. Troubleshooting

Last updated on September 27, 2023 - v0.3

1. Preface

Starting with version 6.6, IDA Pro can debug Android applications written for the Dalvik Virtual Machine. This includes source level debugging too. This tutorial explains how to set up and start a Dalvik debugging session.

2. Installing Android Studio

First of all we have to install the Android SDK from the official site Android Studio.

3. 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.

4. Android Device

Start the Android Emulator or connect to the Android device.

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

5. Installing the App

IDA assumes 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

6. Loading Application into IDA

IDA can handle both .apk app bundles, or just the contained .dex files storing the app's bytecode. If we specify an .apk file, IDA can either extract one of the contained .dex files by loading it with the ZIP load option, or load all classes*.dex files when using the APK loader.

File name			Method	Size
res/layout/activity_display_message.xml			DEFLATED	652
res/layout/activity_main.xml			DEFLATED	804
res/menu/display_message.xml			DEFLATED	464
res/menu/main.xml			DEFLATED	464
AndroidManifest.xml			DEFLATED	2176
resources.arsc			STORED	2680
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				
¢₽ок	X Cancel Sear	ch 🔀 Help		

7. Dalvik Debugger Options

The main configuration of the dalvik debugger happens resides in "Debugger > Debugger Options > Set specific options":

Connection Settings			
ADB executable	/usr/bin/adb	•	
<u>C</u> onnection string		•	
Emulator/device serial number	emulator-5554	•	
Start Application			
<u>Fill from AndroidManifest.xml</u>			
<u>P</u> ackage name	com.example.myfirstapp		•
Activity	com.example.myfirstapp.MainActivity		-
(Alternative) Start Command			-
APK debuggable: true			
Detect Local Variable Types			
○ A <u>l</u> ways			
• Auto			
○ <u>N</u> ever			
Other Options			
show object <u>I</u> D			
✓ preset <u>B</u> PTs			
[OK X Cancel		

7.1. Connection Settings

7.1.1. 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.

7.1.2. Connection string

Specifies the argument to the adb connect command. It is either empty (to let adb figure out a meaningful target) or a <<u>host>[:<port>]</u> combination to connect to a remote device somewhere on the network.

7.1.3. Emulator/device serial number

Serial number of an emulator or a device. Passed to adb's -s option. This option is useful if there are multiple potential target devices running. For the official Android emulator, it is typically emulator-5554.

7.2. Start Application

7.2.1. Fill from AndroidManifest.xml

Press button and point IDA to either the APK or the AndroidManifest.xml file of the mobile app. IDA then automatically fetches the package name and application start activity, as well as the debuggable flag from the specified file.

7.2.2. Package Name

Package name containing the activity to be launched by the debugger.

7.2.3. Activity

Start activity to be launched by the debugger.

7.2.4. Alternative Start Command

Usually IDA builds the start command from the package and activity name and launches the APK from the command line as follows:

am start -D -n '<package>/<activity>' -a android.intent.action.MAIN -c android.intent.category.LAUNCHER

If that does not match your desired debugging setup, you can enter an alternative start command here. Note that you have to provide package and activity as part of the startup command.

7.2.5. APK Debuggable

The value of the debuggable flag, as extracted from the AndroidManifest.xml or the APK. APKs that do not have the debuggable flag set (most do not) cannot be started on unpatched phones. Hence, while this value is false, IDA will display a (silencable) warning when starting a debugging session. To produce a debuggable APK that has the flag set to true, please revert to third-party tooling.

7.3. Detect Local Variable Types

This controls the behavior of IDA's type guessing engine. "Always" and "Never" are pretty self-explanatory: The options force-enable or force-disable type guessing. "Auto" means that type guessing is disabled for Android APIs < 28 and enabled on APIs >= 28. If you work with very old (i.e. API 23 and lower) Android devices and experience crashes during debugging, set this option to "Never". Note that when type guessing is disabled, IDA automatically assumes int for unknown variable types, which causes warnings on API 30 and above.

Local Variables with Type Guessing Deactivated

<u>F</u> ile	<u>E</u> dit Jump Searc <u>h</u> <u>V</u> iew Debugger <u>Options</u> <u>V</u>	<u>/</u> indows Help						
	🚽 🖉 - 🗣 - 📾 🗊 🖂 🖓 - 🔺	• C. D. Q ·	9, - [] 🗳 🔶	🕨 🔲 🔲 Dalvik debug	ger 👻 🖹 ℃	c = =	-1	
								1
	ibrary function 📃 Regular function 📕 Instruction 📗	Data 📕 Unexplo	red 📃 External sy	mbol 🔜 Lumina functio	n			
	Debug View	X	Struct	tures	×]	Enums	×
	IDA View-IP	×	Source view	v: ./MainActivity.java	×	👅 Genera	al registers	- Ø X
•	CODE:0004EE44 .line 22					IP 0004EE	52 🕞 MainActivity	_onCreateOptionsMenu
	CODE:0004EE44 invoke-virtual CODE:0004EE4A move-result-object	v 0		.getMenuInflater()	imp. @ _d			
IP.	CODE:0004EE4C const CODE:0004EE52 invoke-virtual	v1, 0x7F07 {v0, v1, m	0001 enu}, <void men<="" td=""><td>uInflater.inflate(i</td><td>int, ref)</td><td></td><td></td><td></td></void>	uInflater.inflate(i	int, ref)			
	CODE:0004EE58 .line 23 CODE:0004EE58 const/4	v0, 1						
	CODE : 0004EE5A	V0, 1				4		•
•	CODE:0004EE5A locret: CODE:0004EE5A return	vo				🛞 Module	25	0 🗙
	CODE:0004EE5A Method End CODE:0004EE5A #					Path		
	CODE:0004EE5C # ===================================					😹 <jdwp< td=""><td>process></td><td></td></jdwp<>	process>	
•	CODE:0004EE5C # Method 5099 (0x13eb) CODE:0004EE5C word 4EE5C:.short 6	#	DATA XREF: CODE	:000898F7+i				
•	CODE:0004EE5C CODE:0004EE5E .short 2		Number of regis	ters : 0x6 rgs (in words) : 0x	-2	4		•
	CODE:0004EE60 .short 3	#	Size of output	args (in words) : 0				
	CODE:0004EE62 .short 0 CODE:0004EE64 .int 0x809EE		Number of try_i Debug info	tems : 0x0		🛃 Thread	s	
•	CODE:0004EE68 .int 0x20	÷		le (in 16-bit units)	: 0x20	Decimal	Hex State	Name
	CODE:0004EE6C # Source file: MainActivity CODE:0004EE6C public void com.example.myf		ivity.sendMessa	ige (1	1 Ready	<jdwp proces<="" td=""></jdwp>
	CODE:0004EE6C android.view.View vie CODE:0004EE6C this = v4	w)				€ 10€ 9	A Ready 9 Ready	<10> Binder_2
C	CODE-0004EE6C view = v5 004EE52 0004EE52: MainActivity onCreateOption	Menu@7L+F (Sunc)	bronized with TD			€ 9 € 8	9 Ready 8 Ready	<9> Binder_1 <8> FinalizerWa
4		SHendernin (Syne)	aronized with it		Þ			•
Q	Hex View-1		×	8		Loca	ls	×
Name		Туре				Location		
▶ th	. , . ,					v2		
> m	enu {mActionItems=,mCallback=,mContext=,mCurre Bad type	ntmenuinto com	.android.internal.vi	ew.menu.menuBuilder *		v3 v0		
v1	<u>, , , , , , , , , , , , , , , , , , , </u>					vl		
	nput rrrr. chreau has scarceu (cru-o) voz Rerei	encequeuebaemor	•					0 8
FFFF	FFFF: thread has started (tid=5) <5> Compi FFFF: thread has started (tid=3) <3> Signa	ler						-
FFFF	FFFF: thread has started (tid=2) <2> GC							
	FFFF: thread has started (tid=11) <11> Asy FFFF: thread has started (tid=12) <12> Asy							
FFFF	FFFF: thread has started (tid=13) <13> Bir on>ida_kernwin.set_dock_pos("Locals", "Hey	der_3	ernwin DD TAD					
True		view-1", ida_k	ernwin.br_TAB)					•
ID	c							
AU:	idle Down Disk: 169GB							

Local Variables with Type Guessing Activated

	Debug View	×	0	Structures	×	E		Enums		
	IDA View-IP	×		Source view: ./MainActi		×	👅 General registe			
* C C C	ODE:0004EE44 .line 22 ODE:0004EE44 invoke-virtual ODE:0004EE4A move-result-object ODE:0004EE4C const CDE:0004EE425 invoke-virtual ODE:0004EE55 .line 23	v			inflater() imp.	9 _d	IP 0004EE52 🕞 Ma	ainActivity_	onCreateOption	nsM
С	ODE:0004EE58 const/4	v	0, 1				4			
	ODE:0004EE5A ODE:0004EE5A locret:						Modules			_
	ODE:0004EE5A return ODE:0004EE5A Method End	v	0						0	Ø
С	ODE:0004EE5A #						Path			
0	ODE:0004EE5C # ===================================	eb)	#	DATA XREF: CODE:000898F7			Solution of the second seco	>		
	ODE:0004EE5C ODE:0004EE5E .short 2			Number of registers : 0x Size of input args (in w			•			
	ODE:0004EE60 .short 3 ODE:0004EE62 .short 0		#	Size of output args (in	words) : 0x3					
С	ODE:0004EE64 .int 0x809EE ODE:0004EE68 .int 0x20		#	Number of try_items : 0x Debug info Size of bytecode (in 16-			📑 Threads			ð
0	ODE:0004EE6C android.view. ODE:0004EE6C this = v4 ODE:0004EE6C view = v5 4EE52 0004EE52: MainActivity_onCrea		L+E (Syncl	hronized with IP)			 ■ 102 ● 66 ■ 103 ● 67 ■ 104 ● 68 ■ 	Ready Ready Ready	Signal Catch ADB-JDWP C Jit thread po	Со
	Hex Vie	ew-1		×			Locals			
ne	Value		Туре	•		I	ocation			
his	{mActionBar=,mActionModeTypeStartir						v2			
neni /0	u {mActionItems=,mCallback=,mContext {mActionProviderConstructorArgument:			.android.internal.view.menu.Me	enuBuilder *		v3 v0			
/1	0x7F070001	s={},mactionview	/00				v0 v1			
Outp	out									ð
FFF	<pre>rr. cnicau has started (tid=100) FF: thread has started (tid=110) FF: thread has started (tid=111) FF: thread has started (tid=112) FF: thread has started (tid=113)</pre>	Binder:4566_2 Binder:4566_3 Profile Saver								

7.4. Other Options

7.4.1. Show object ID

If active, IDA shows the object ID assigned by the Java VM for composite (non-trivial) types in the local variables window.

7.4.2. Preset BPTs

If active, IDA sets breakpoints at the beginning of all (non-synthetic, non-empty) methods of the start activity class specified in the Activity field above.

8. 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.

9. Setting 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:

ame	Address	Public	
f ViewDragHelper.setDragState@VI	0004E9D4	Р	
f ViewDragHelper.setEdgeTrackingEnabled@VI	0004EA08	Р	
f ViewDragHelper.setMinVelocity@VF	0004EA20	Р	
f ViewDragHelper.settleCapturedViewAt@ZII	0004EA38	Р	
f ViewDragHelper.shouldInterceptTouchEvent@ZL	0004EA90	Р	
🕖 ViewDragHelper.smoothSlideViewTo@ZLII	0004EC90	Р	
f ViewDragHelper.tryCaptureViewForDrag@ZLI	0004ECB8	Р	
🕖 BuildConfigclinit_@V	0004ED00	Р	
f BuildConfiginit_@V	0004ED24	Р	
🚺 DisplayMessageActivityinit_@V	0004ED3C	Р	
f DisplayMessageActivity.onCreate@VL	0004ED54	Р	
f DisplayMessageActivity.onCreateOptionsMenu@ZL	0004EDA0	Р	
f DisplayMessageActivity.onOptionsItemSelected@ZL	0004EDC8	Р	
🚺 MainActivityinit_@V	0004EE08	Р	
🚹 MainActivity.onCreate@VL	0004EE20	Р	
🚺 MainActivity.onCreateOptionsMenu@ZL	0004EE44	Р	
🚺 MainActivity.sendMessage@VL	0004EE6C	Р	
🚺 R\$attrinit_@V	0004EEBC	Р	
🚺 R\$dimeninit_@V	0004EED4	Р	
🚺 R\$drawableinit_@V	0004EEEC	Р	
🚺 R\$idinit_@V	0004EF04	Р	
🚺 R\$layoutinit_@V	0004EF1C	Р	
🚺 R\$menuinit_@V	0004EF34	Р	
🚺 R\$stringinit_@V	0004EF4C	Р	
🚺 R\$styleinit_@V	0004EF64	Р	
🚺 Rinit_@V	0004EF7C	Р	
10 empty_str	00052BAD		
ine 10138 of 16151, /MainActivity.onCreate@VL	00050000		

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

10. 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:

L/LESLS/	ui/iua	scieensi	013-1	msc/c	asses.ue	^	
				No de	bugger		
₽₽₀ - []	l 👍 📢	> 🕨 🗖		Dalvik	debugger	10	c C
ored	Extern	al symbol	Lu	ımina f	unction		
es 🗙	Ξ	Enums	X	M	Imports	X	
0 AB136 7F2F3	CDD 904	185E4EE3B	4D4D	DDC80			<u>,</u>
		erface ve		n 8			
t Dex	File v	version 3	5				

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

- Launch a new copy of the application (Start process)
- Attach to a running process (Attach to process)

10.1. Launching the App

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.

10.2. Attaching to a Running App

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.

ID	Name
2294	mple.myfirstapp
Line 1 of 1	
	🖉 OK 🛛 🗶 Cancel 🛛 Search 🛛 👯 Help

We just select the process we want to attach to.

11. 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 > 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

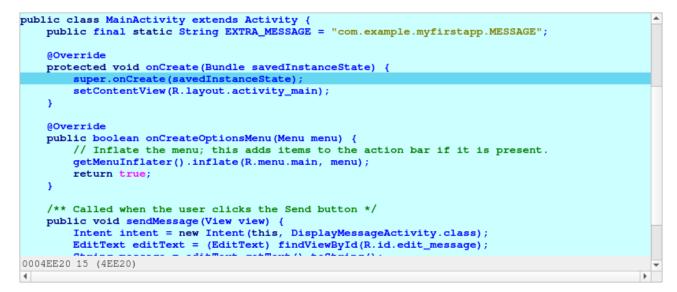
"Cancel" button, quite often we may find ourselves outside of the application, in the system code. The value of the IP register is 0xFFFFFFFF in this case, and stack trace shows only system calls and a lot of 0xFFFFFFFF. It means that IDA could not locate the current execution position inside the application. 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 0xFFFFFFF.

11.1. 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.



Perform "Step over" action (the hot key is <F8>) two times and open the "Locals" window, we will see something like the following:

Value	Туре	Location
{mActionBar=,mActivityInf	com.example.myfirstapp.M	v1
null	android.os.Bundle *	v2
Bad type		v0
	{mActionBar=,mActivityInf	{mActionBar=,mActivityInf com.example.myfirstapp.M null android.os.Bundle *

If information about the frame is available (the symbol table is intact) or type guessing is enabled 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 and query them with an explicit type (see below).

11.2. Watch View Window

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

Name	Value	Туре	Location
(int)v0	0x7F030001	int	v0

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.

IMPORTANT

Attention! On Android API versions 23 and below 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.

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.

Overall we recommend to debug on a device that runs at least Android API 24.

12. Troubleshooting

- · 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.
- Android APIs 24 and 25 are known to return wrong instruction sizes during single stepping. Try migrating to a different Android API if you have trouble with single steps.
- IDA exposes a subset of the JDWP specification as IDC commands. (Usually the name from the specification prefixed with JDWP_).
- Android APIs 23 and below crash if type guessing is enabled. Remedy this by setting the Detect Local Variable Types option to Never or migrate to a newer Android API.