Android Debugging

v. 0.4

This document describes utilities, tips and best practices for improving debugging for any application on Android platforms. Most of these tips apply to both native and Unity applications.

1. Adb

Android Debug Bridge (adb) is included in the Android SDK and is the main tool used to communicate with an Android device for debugging. We recommend familiarizing yourself with it by reading the official documentation located here:
http://developer.android.com/tools/help/adb.html

1.1 Using adb

Using adb from the OS shell, it is possible to connect to and communicate with an Android device either directly through USB, or via TCP/IP over a WIFI connection. The Android Software Development Kit and appropriate device drivers must be installed before trying to use adb (see *Device and Environment Setup Guide* for more information).

To connect a device via USB, plug the device into the PC with a compatible USB cable. After connecting, open up an OS shell and type:

```
adb devices
```

If the device is connected properly, adb will show the device id list such as:

```
List of devices attached
ce0551e7                device
```

Adb may not be used if no device is detected. If your device is not listed, the most likely problem is that you do not have the correct Samsung USB driver - see *Device and Environment Setup Guide* for more information. You may also wish to try another USB cable and/or port.

```
- waiting for device -
```

Note that depending on the particular device, detection may be finicky from time to time. In particular, on some devices, connecting while a VR app is running or when adb is waiting for a device, may prevent the device from being reliably detected. In those cases, try ending the app and stop adb using Ctrl-C before reconnecting the device. Alternatively the adb service can be stopped using the following command after which the adb service will be automatically restarted when executing the next command:

```
adb kill-server
```

Multiple devices may be attached at once, and this is often valuable for debugging client/server applications. Also, when connected via WIFI and also plugged in via USB, adb will show a single device as two devices. In the multiple device case adb must be told which device to work with using the `-s` switch. For example, with two devices connected, the `adb devices` command might show:

```
List of devices attached
ce0551e7                device
10.0.32.101:5555        device
```

The listed devices may be two separate devices, or one device that is connected both via WIFI and plugged into USB (perhaps to charge the battery). In this case, all adb commands must take the form:

```
adb -s <device id> <command>
```

where <device id> is the id reported by adb `devices`. So, for example, to issue a logcat command to the device connected via TCP/IP:

```
adb -s 10.0.32.101:55555 logcat -c
```

and to issue the same command to the device connected via USB:

```
adb -s ce0551e7
```

1.2 Connecting adb via WIFI

Connecting to a device via USB is generally faster than using a TCP/IP connection, but a TCP/IP connection is sometimes indispensable, especially when debugging behavior that only occurs when the phone is placed in the Gear VR, in which case the USB connection is occupied by the Gear VR jack.

To connect via TCP/IP, first determine the IP address of the device and make sure the device is already connected via USB. You can find the IP address of the device under *Settings* -> *About device* -> *Status*. Then issue the following commands:

```
adb tcpip <port>
adb connect <ipaddress>:<port>
```

For example:

```
> adb tcpip 5555
restarting in TCP mode port: 5555
> adb connect 10.0.32.101:5555
connected to 10.0.32.101:5555
```

The device may now be disconnected from the USB port. As long as `adb devices` shows only a single device, all adb commands will be issued for the device via WIFI.

To stop using the WIFI connection, issue the following adb command from the OS shell:

```
adb disconnect
```

2. Logcat

The Android SDK provides the logcat logging utility, which is essential for determining what an application and the Android OS are doing. To use logcat, connect the Android device via USB or WIFI, launch an OS shell, and type:

```
adb logcat
```

If the device is connected and detected, the log will immediately begin outputting to the shell. In most cases, this raw output is too verbose to be useful. Logcat solves this by supporting filtering by tags. To see only a specific tag, use:

```
adb logcat -s <tag>
```

This example:

```
adb logcat -s VrApp
```

will show only output with the “VrApp” tag.

In the native VRLib code, messages can generally be printed using the `LOG()` macro. In most source files this is defined to pass a tag specific to that file. Log.h defines a few additional logging macros, but all resolve to `calling __android_log_print()`.

2.1 Using Logcat to Determine the Cause of a Crash

Logcat will not necessarily be running when an application crashes. Fortunately, it keeps a buffer of recent output, and in many cases a command can be issued to logcat immediately after a crash to capture the log that includes the backtrace for the crash:

```
adb logcat > crash.log
```

Simply issue the above command, give the shell a moment to copy the buffered output to the log file, and then end adb (Ctrl+C in a Windows cmd prompt or OS X Terminal prompt). Then search the log for “backtrace:” to locate the stack trace beginning with the crash.

If too much time as elapsed and the log does not show the backtrace, there a full dump state of the crash should still exist. Use the following command to redirect the entire dumpstate to a file:

```
adb shell dumpstate > dumpstate.log
```

Copying the full dumpstate to a log file usually takes significantly longer than simply capturing the currently buffered logcat log, but it may provide additional information about the crash.

2.2 Getting a Better Stack Trace

The backtrace in a logcat capture or dumpstate generally shows the function where the crash occurred, but does not provide line numbering. To get more information about a crash, the Android Native Development Kit (NDK) must be installed. When the NDK is installed, the ndk-stack utility can be used to parse the logcat log or dumpstate for more detailed information about the state of the stack. To use ndk-stack, issue the following:

```
ndk-stack -sym <path to symbol file> -dump <source log file> >
stack.log
```

For example, this command:

```
ndk-stack -sym VrNative\Oculus360Photos\obj\local\armeabi-v7a -dump
crash.log > stack.log
```

uses the symbol information for Oculus 360 Photos to output a more detailed stack trace to a file named stack.log, using the backtrace found in crash.log.

3. Performance Profiling

See *Performance Guidelines* for information specific to performance optimization and profiling.

Last Update: Nov 10, 2014