Explaining Monkop Reports

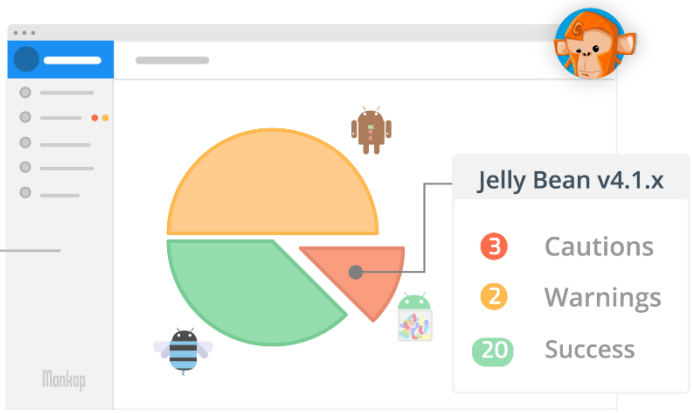
Dashboard & Report Sections

Version 1.2

Contents

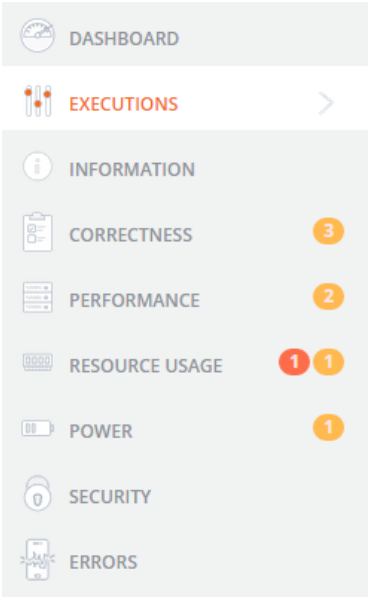
Overview

This document explains how to understand a Monkop report after each execution (test-ride). This is what the report looks like:



The report is a web HTML page stored in the cloud, therefore, it's accessible from any browser. It contains a left navigation panel and a dashboard overview.

Navigation Panel: The left menu contains all the sections and highlighted warnings and cautions. This panel enables users to navigate across:



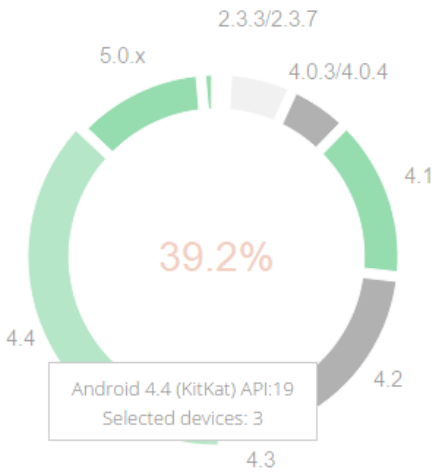
- **Dashboard**: Contains market stats and selected devices
- **Executions**: The most important section, containing each **device's information** such as OS version, screen, and hardware details. It also contains **execution-related information** such as: videos, screenshots, installation process, timing measures, resources and logs.
- **Information**: General information about the application under test such as: version, compatibility, permissions requested, activities and included libraries.
- **Correctness, Performance, Resource Usage, Security** and **Power** sections: each contains analyzed behavior across all devices. Yellow / red indicators are values out of Monkop's default thresholds, which are based on Google's best practices and important market research findings.
- **Errors**: Contains crash reports and logs when the app is not responding (ANR).

Dashboard

This section contains a market chart, execution details, a device selection segment (by screen density), and test-run information.

The following chart shows the latest Android market stats based on data collected by Google.

Android Market (% API level)



The chart shows information about selected devices when moused-over. Reference:

One or more selected devices with that Android version installed.

No device selected. Reason: not selected by user / no license / device not found.

Android version not supported by the app.

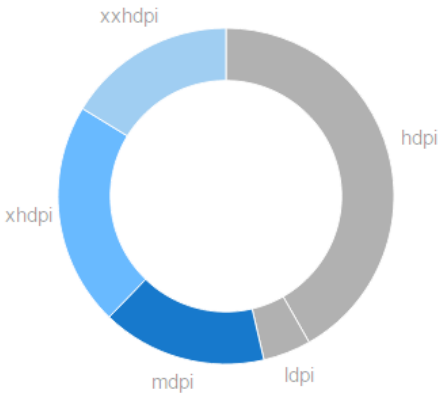
Bellow, users will find the summary about Monkop's executions results:

Executions

Ride	Device	OS	Size/Density	Execution	Startup
●	Google Nexus 4	5.0.x (API:21)	normal/xhdpi	30min 02sec 7ms	7ms
●	Google Nexus 7	5.0.x (API:21)	large/xhdpi	30min 13sec	6ms
●	Samsung Galaxy Note 3 (SM-N900V)	4.4 (API:19)	normal/xxhdpi	30min 12sec 5ms	01sec 8ms
●	Samsung Galaxy Note 10.1" (GT-N8013)	4.1 (API:16)	xlarge/mdpi	30min 25sec 9ms	01sec 7ms
●	Samsung Galaxy S III (GT-I9300)	4.3 (API:18)	normal/xhdpi	30min 13sec 4ms	01sec 7ms
●	LG G2 (LS980)	5.0.x (API:21)	normal/xxhdpi	30min 12sec 2ms	6ms
●	Google Nexus 10	5.1 (API:22)	xlarge/xhdpi	30min 10sec 4ms	5ms
●	Motorola G (XT1031)	4.4 (API:19)	normal/xhdpi	30min 11sec 3ms	7ms
●	ZTE Zinger (Z667T)	4.4 (API:19)	normal/mdpi	30min 01sec 8ms	01sec 2ms

Execution: This column shows how many minutes the monkey (crawler) has played with the app.
Startup is the amount of time taken by the app to become responsive to the user, an important value.

Device selection (Screen Density DPI)



Test run

Devices:	9
Total time:	04hs 31min 43sec 6ms

Executions

This is the **most important section**, since it has each device-specific execution. It shows executions that the user has manually selected, or those that Monkop has automatically selected.

Google Nexus 7

Android 5.0.2

Details

Google Nexus 4

Android 5.0.1

Details

Google Nexus 10

Android 5.1.1

Details

LG G2 (LS980)

Android 5.0.2

Details

Motorola G (XT1031)

Android 4.4.4

Details

Samsung Galaxy S III (GT-I9300)

Android 4.3

Details

Samsung Galaxy Note 10.1" (GT-N8013)

Android 4.1.2

Details

Samsung Galaxy Note 3 (SM-N900V)

Android 4.4.2

Details

ZTE Zinger (Z667T)

Android 4.4.2

Details

By clicking on any device, users will find device specific information about executions.

Samsung Galaxy S III (GT-I9300)

Android version:	4.3	Screen orientation:	port
Manufacturer:	samsung	Screen resolution:	720x1280
Model:	GT-I9300	Layout size:	Normal
CPU Architecture:	armeabi-v7a	Display density:	320dpi (xhdpi)
Dalvik heap size limit:	64MB	OpenGL ES:	2.0
Dalvik large heap size limit:	256MB		

Ride

APK	Install	Process	Uninstall

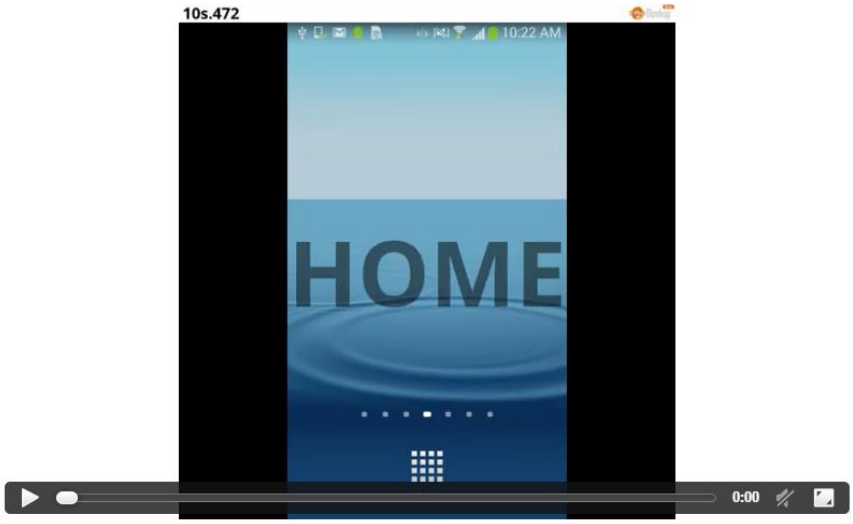
Execution

Time Elapsed:	30min 13sec 4ms (100% of expected time)
Application startup time:	01sec 7ms

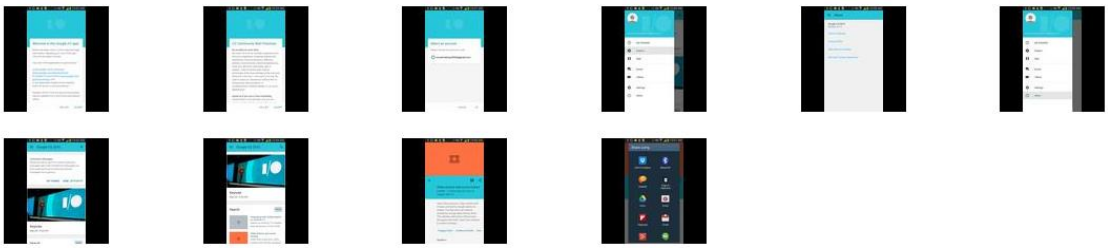
Below this information, users will find details about device performance, videos, screenshots and logs.

View

Exploration



Forms



Resources

- **CPU**
- **Threads**
- **Memory**(Pss, Dalvik/Native Heap, GC usage)
- **Network**(Transmit/Receive, Adress List)
- **Render**(FPS, Render Time, Janks)
- **Database**(SQL Time, TOP SQL's)
- **Power usage**(Energy consumption)

Detailed performance information is stored in these sections. Clicking on each header and drilling-down, users will find charts and performance spikes.

Information

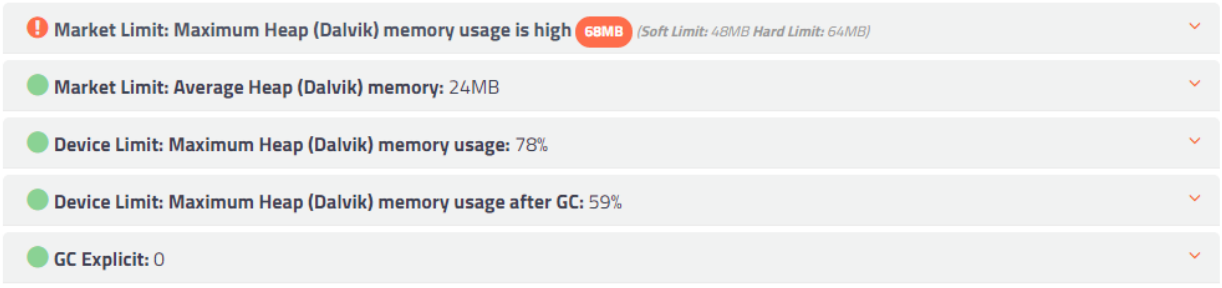
Here users will find:

- General information about the app such as version, package name, launch activity name, and if the use of large memory heap is enabled.
- Compatibility: min and max android APLI levels defines app compatibility in user's phones depending on OS version, also screen sizes / densities and localization options.
- Libraries: Monkop analyzes the app looking for known libraries and their versions. This enables Monkop to evaluate potential vulnerabilities and deprecated / unsupported libraries that are part of the app.
- Permissions: Allow detailed permissions that will be requested to govern the app capabilities.
- Activities: All activities (screen app component) created inside the app.
- Receivers: Declared intents to be received by other apps / system broadcasts.
- Services: Operations that will run without UI.

Indicators

The following five sections in the document are committed to analyzing behavior across all devices selected for the test-ride. The yellow/red indicator bubbles are **important items evaluated out of default thresholds** set by Monkop, which are based on Google's best practices for Android development and other market research findings.

Typically, indicator thresholds contain two limits: a soft limit and a hard limit. This defines the color (yellow for reaching the soft-limit, and red for the hard-limit). Indicators are organized in related-topics, for example, the following are indicators related to memory:



Here, the app seems to be using too much memory, so now, the user will need to click on the caution indicator to be able to understand:

- Which devices are experiencing this?
- How was the app UI in that moment?
- How was the user interaction with the UI seconds before and after?
- The exact instant time-stamp of the execution.

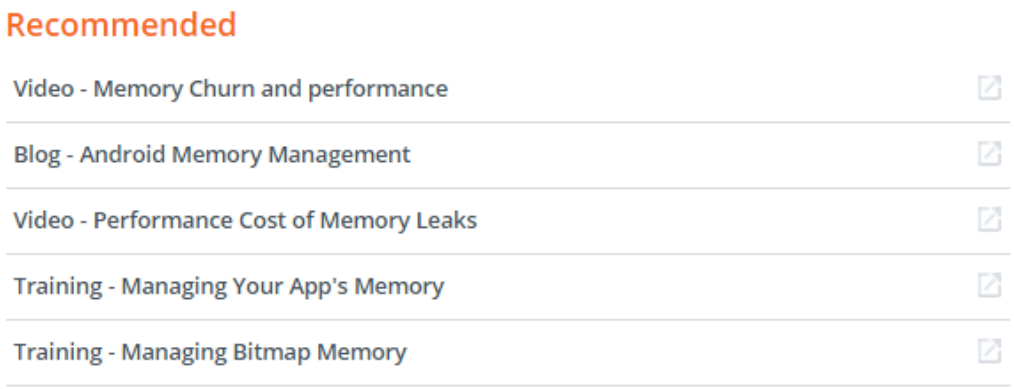
This can be achieved by looking at **indicator details** and related **events** sub-section presented below each one. Also, it is important to **cross-reference the information** with device-specific behavior presented in the previous section, "Execution."

Device	Value
Motorola G (XT1031) (4.4.4):	41MB
Samsung Galaxy S III (GT-I9300) (4.3):	⚠ 50MB
LG G2 (LS980) (5.0.2):	19MB
Google Nexus 10 (5.1.1):	10MB
Lenovo A369i (4.2.2):	33MB
ZTE Zinger (Z667T) (4.4.2):	16MB
Google Nexus 4 (5.0.1):	⚠ 52MB
Acer Iconia A1-810 7.9" (4.2.2):	17MB
HTC One (M8) (4.4.2):	❗ 68MB
Google Nexus 7 (5.0.2):	⚠ 52MB

Events

Device	Activity	Screen	Time	Value	Preview
Google Nexus 7 (5.0.2)	com.google.samples.apps.iosched.map.MapActivity		13m4s.674	⚠ 49MB	Play
HTC One (M8) (4.4.2)	android/com.android.internal.app.ChooserActivity		17m28s.756	❗ 68MB	Play

Additionally, after each indicator, users will find a list of recommended links to consider reading.



Correctness

Correctness is the first of the five sections committed to analyzing behavior across all devices selected on the ride.

The following are analyzed in the “Correctness” section:

Activities

Android Activities are one of the most important parts of an application's overall lifecycle. The way activities are launched and how developers manage all of them together is a fundamental part of the platform's application model.

In order to improve performance, developers should try to provide a user interface that avoids creating unnecessary activities and excessively high levels of consumption.

Activities amount

All non-trivial Android applications are made up of a number of different functional screens and hence, multiple activities. Although multiple screens allow for building complex applications, they also require careful management. In particular, developers need to deal with activities that are no longer visible since Android OS will place them into the background and may terminate activities that are not used for a period of time. The use of multiple activities also requires careful consideration of the interaction and navigation model that the user will experience.

Activities stack

Creating multiple activities causes Android to put them into the 'Back Stack' in order to save states such as text form, scroll position, and other data. Multiple tasks can be held in the background at once. However, if the user is running many background tasks at the same time, the system might begin destroying background activities in order to recover memory, causing the activity states to be lost.

Layout

Layouts are a key part of Android applications that directly affect the user experience. A poorly implemented layout can lead to a memory hungry application with slow UIs.

UI controls

Control count for each screen.

Nested layouts

It is a common misconception that using the basic layout structures leads to the most efficient layouts. However, each widget and layout added to an application requires initialization, layout, and drawing.

Libraries

External library code is not often written for mobile environments and can be inefficient when used for a mobile client. At the very least, when deciding to use an external library, one should assume that one is taking on a significant porting and maintenance burden to optimize the library for mobile. Plan for that work up-front and analyze the library in terms of code size and RAM footprint before deciding to use it at all.

Deprecated libraries

If deprecated libraries are shown in this section, it means that libraries are being used that aren't up to date. On the other hand, abandoned libraries are those without support or have pending bugs from some time ago. Usually, library updates are important in order to improve security, performance, compatibility & bug fixes.

Performance

Performance is the section in which users will find behavior analyses on the **app's performance on the device**. As in the previous section, each yellow / red indicator bubbles are **important indicators to look over**, based on default thresholds set by Monkop.

The following are analyzed in the "Performance" section:

Application Process

An Android application is a single installable unit that can be started and used independently of other applications.

Every application could have one application class, which is instantiated as soon as the application starts and is the last component stopped during shutdown.

An app must declare its components in AndroidManifest.xml ('manifest' file). There are four different types of components: Activities, Services, Content Providers and Broadcast Receivers. Each type serves different purposes and has its own lifecycle that defines how the component is created and destroyed. When the system starts a component, it starts the process for that app (if it's not already running) and instantiates the classes needed for the component.

Launch time

Memory usage and launch time are some of a developer's top performance concerns. This is rightfully so because users expect apps to load in just three seconds. If apps don't launch fast enough, they may lose a significant amount of users.

Render

When building an application, it's important to consider exactly what the graphical demands will be. Varying graphical tasks are best accomplished with varying techniques. For example, graphics and animations for a rather static application should be implemented much differently than graphics and animations for an interactive game. No matter what type of application it is, there are certain situations that affect user experience (response rate, fluency, use of resources, battery etc.). Times drawn possibly reflect some things are not performing in the best possible way for what they were intended to do.

Render Time

To achieve fluid rendering (60 fps), each frame must be completed in less than 16ms. If not, the application creates a disruption in the animation and sometimes freezes. Also, elevated drawing to the screen needs high CPU and/or GPU usage in order to maintain a constant rate, causing battery drain.

Resource Usage

This section is the preferred one for performance lovers, because it contains **CPU and memory performance** on each device selected.

The following are analyzed in the “Resource Usage” section:

CPU

The CPU is the unit responsible for carrying out all instructions of an application and all the necessary instructions for running different subsystems that keep Android OS running (multimedia, audio, render, etc.).

CPU usage

When the CPU usage is high, the user may experience app sluggishness or higher battery usage (among other symptoms). Since the CPU usage is a shared resource, abusing it may prevent other running services from working correctly, affecting the overall Android user experience. With higher levels of instruction, the CPU increases its speed with a consequent increase in use of voltage that causes the battery to drain faster.

Memory

Random-access memory (RAM) is one of the most valuable resources in any software development environment, but it's even more valuable for several mobile operating systems where physical memory is constrained.

Memory usage

To maintain a functional multi-tasking environment, Android sets a fixed limit on the Dalvik heap size for each app. The exact Dalvik heap size limit varies across devices, based on how much RAM the device has available overall. If an app has reached the heap capacity and tries to allocate more memory, it will receive an OutOfMemoryError.

Network

Using the wireless radio to transfer data is potentially one of an app's most significant sources of extra fees, poor user experience and battery drain. To minimize the associated effects on network activity, it's important to understand how the connectivity model will affect the underlying radio hardware.

Network usage

If the application performs a lot of network operations, one should provide user settings that allow users to control the app's data habits, such as how often it syncs data, whether to perform uploads/downloads only when connected to Wi-Fi, whether to use data while roaming, and so on.

Power

Power consumption **depends a lot on how the monkey(crawler) has used the app**, so it is important to **take a look at device-specific** information about power consumption, in the "**Execution"** section. The "Power" section only shows a "power-score" based on average behavior.

This score is based on AT&T research and public publications on ARO project.

Power usage is Medium

Errors

This section enables users to avoid hard log-diving. When an app crashes, it throws an exception or doesn't respond. All detailed information extracted from much of the app's logs and device information will be presented here:

Application not responding (ANR)

If an app stops responding, users get a dialog that allows them to wait or close the app. When these dialogs appear, they're known as 'Application not responding' errors or ANRs.

Android will display the ANR dialog when it detects one of the following conditions:

- No response to an input event (such as key press or screen touch events) within 5 seconds.
- A Broadcast Receiver hasn't finished executing within 10 seconds.

ANR Dump Logs

When an app stops responding (ANR), Android generates dump files containing CPU and Threads information. This enables devs to identify CPU usage on each process at the moment when the app froze and provides a snapshot containing threads information (thread, mutex and stack information).

Crash

An application typically crashes when it performs an operation that it's not allowed to do by the operating system. The operating system then triggers an exception or signal in the application.

Exceptions

An exception is an event that occurs during the execution of a program that disrupts the normal flow of instructions. An exception contains a snapshot of the execution stack of its thread at the time it was created. It can also contain a message string that gives more information about the error.