ART and Dalvik

Android runtime (ART) is the managed runtime used by applications and some system services on Android. ART and its predecessor Dalvik were originally created specifically for the Android project. ART as the runtime executes the Dalvik Executable format and Dex bytecode specification.

ART and Dalvik are compatible runtimes running Dex bytecode, so apps developed for Dalvik should work when running with ART. However, some techniques that work on Dalvik do not work on ART. For information about the most important issues, see Verifying App Behavior on the Android Runtime (ART) (http://developer.android.com/guide/practices/verifying-apps-art.html).

IN THIS DOCUMENT

- ART Features
  - Ahead-of-time (AOT) compilation
  - Improved garbage collection
  - Development and debugging improvements
- Reporting Problems

ART Features

Here are some of the major features implemented by ART.

Ahead-of-time (AOT) compilation

ART introduces ahead-of-time (AOT) compilation, which can improve app performance. ART also has tighter install-time verification than Dalvik.

At install time, ART compiles apps using the on-device **dex2oat** tool. This utility accepts DEX (http://source.android.com/devices/tech/dalvik/dex-format.html) files as input and generates a compiled app executable for the target device. The utility should be able to compile all valid DEX files without difficulty. However, some post-processing tools produce invalid files that may be tolerated by Dalvik but cannot be compiled by ART. For more information, see Addressing Garbage Collection Issues (http://developer.android.com/guide/practices/verifying-apps-art.html#GC_Migration).

Improved garbage collection

Garbage collection (GC) can impair an app's performance, resulting in choppy display, poor UI responsiveness, and other problems. ART improves garbage collection in several ways:

- One GC pause instead of two
- Parallelized processing during the remaining GC pause
- Collector with lower total GC time for the special case of cleaning up recently-allocated, short-lived objects
- Improved garbage collection ergonomics, making concurrent garbage collections more timely, which makes `GC_FOR_ALLOC` events extremely rare in typical use cases
- Compacting GC to reduce background memory usage and fragmentation

Development and debugging improvements

ART offers a number of features to improve app development and debugging.

Support for sampling profiler

Historically, developers have used the Traceview (http://developer.android.com/tools/help/traceview.html) tool (designed for tracing application execution) as a profiler. While Traceview gives useful information, its results on Dalvik have been skewed by the per-method-call overhead, and use of the tool noticeably affects run time performance.

ART adds support for a dedicated sampling profiler that does not have these limitations. This gives a more accurate view of app execution without significant slowdown. Sampling support was added to Traceview for Dalvik in the KitKat release.

Support for more debugging features

ART supports a number of new debugging options, particularly in monitor- and garbage collection-related functionality. For example, you can:

- See what locks are held in stack traces, then jump to the thread that holds a lock.
- Ask how many live instances there are of a given class, ask to see the instances, and see what references are keeping an object live.
- Filter events (like breakpoint) for a specific instance.
- See the value returned by a method when it exits (using “method-exit” events).
- Set field watchpoint to suspend the execution of a program when a specific field is accessed and/or modified.

Improved diagnostic detail in exceptions and crash reports

ART gives you as much context and detail as possible when runtime exceptions occur. ART provides expanded exception detail for `java.lang.ClassCastException` (http://developer.android.com/reference/java/lang/ClassCastException.html), `java.lang.ClassNotFoundException` (http://developer.android.com/reference/java/lang/ClassNotFoundException.html), and `java.lang.NullPointerException` (http://developer.android.com/reference/java/lang/NullPointerException.html). (Later versions of Dalvik provided expanded exception detail for `java.lang.ArrayIndexOutOfBoundsException` (http://developer.android.com/reference/java/lang/ArrayIndexOutOfBoundsException.html) and `java.lang.ArrayStoreException` (http://developer.android.com/reference/java/lang/ArrayStoreException.html), which now include the size of the array and the out-of-bounds offset, and ART does this as well.)

For example, `java.lang.NullPointerException` (http://developer.android.com/reference/java/lang/NullPointerException.html) now shows information about what the app was trying to do with the null pointer, such as the field the app was trying to write to, or the method it was trying to call. Here are some typical examples:

```
java.lang.NullPointerException: Attempt to write to field 'int
android.accessibilityservice.AccessibilityServiceInfo.flags' on a null object
reference
```

```
java.lang.NullPointerException: Attempt to invoke virtual method
'java.lang.String java.lang.Object.toString()' on a null object reference
```

ART also provides improved context information in app native crash reports, by including both Java and native stack information.

Reporting Problems

If you run into any issues that aren’t due to app JNI issues, please report them via the Android Open Source Project Issue Tracker at http://b.android.com (http://b.android.com). Please include an `"adb bugreport"` and link to the app in Google Play store if available. Otherwise, if possible, attach an APK that reproduces the issue. Please note that issues (including attachments) are publicly visible.