Getting Started with Swift on Android

The Swift stdlib can be compiled for Android armv7, x86_64, and aarch64 targets, which makes it possible to execute Swift code on a mobile device running Android or an emulator. This guide explains:

1. How to run a simple "Hello, world" program on your Android device.
2. How to run the Swift test suite on an Android device.

If you encounter any problems following the instructions below, please file a bug using https://bugs.swift.org/.

FAQ

Let's answer a few frequently asked questions right off the bat:

Does this mean I can write Android applications in Swift?

No. Although the Swift compiler is capable of compiling Swift code that runs on an Android device, it takes a lot more than just the Swift stdlib to write an app. You'd need some sort of framework to build a user interface for your application, which the Swift stdlib does not provide.

Alternatively, one could theoretically call into Java interfaces from Swift, but unlike as with Objective-C, the Swift compiler does nothing to facilitate Swift-to-Java bridging.

Prerequisites

To follow along with this guide, you'll need:

1. A Linux environment capable of building Swift from source, preferably Ubuntu 18.04 or Ubuntu 16.04. Before attempting to build for Android, please make sure you are able to build for Linux by following the instructions in the Swift project README.
2. The latest version of the Android NDK (r23b at the time of this writing), available to download here: https://developer.android.com/ndk/downloads/index.html.
3. An Android device with remote debugging enabled or the emulator. We require remote debugging in order to deploy built stdlib products to the device. You may turn on remote debugging by following the official instructions: https://developer.chrome.com/devtools/docs/remote-debugging.

"Hello, world" on Android

1. Building the Swift stdlib for Android

Enter your Swift directory, then run the build script, passing the path to the Android NDK:

```
$ NDK_PATH=path/to/android-ndk-r23b
$ utils/build-script \
    -R \                                    # Build in ReleaseAssert mode.
    --android \                             # Build for Android.
    --android-ndk $NDK_PATH \               # Path to an Android NDK.
    --android-arch armv7 \                  # Optionally specify Android
architecture, alternately aarch64 or x86_64
    --android-api-level 21                  # The Android API level to target.
Swift only supports 21 or greater.
```

2. Compiling `hello.swift` to run on an Android device

Create a simple Swift file named `hello.swift`:

```
print("Hello, Android")
```

Then use the built Swift compiler from the previous step to compile a Swift source file, targeting Android:

```
$ NDK_PATH="path/to/android-ndk-r23b"
$ build/Ninja-ReleaseAssert/swift-linux-x86_64/bin/swiftc \          # The Swift
compiler built in the previous step
                                                                     # The location of
the tools used to build Android binaries
    -tools-directory ${NDK_PATH}/toolchains/llvm/prebuilt/linux-x86_64/bin/ \
    -target armv7-unknown-linux-androideabi21 \                      # Targeting
Android armv7 at API 21
    -sdk ${NDK_PATH}/toolchains/llvm/prebuilt/linux-x86_64/sysroot \ # The SDK is the
Android unified sysroot and the resource-dir is where you just built the Swift stdlib.
    -resource-dir build/Ninja-ReleaseAssert/swift-linux-x86_64/lib/swift
    hello.swift
```

This should produce a `hello` executable in the directory you executed the command. If you attempt to run this executable using your Linux environment, you'll see the following error:

```
cannot execute binary file: Exec format error
```

This is exactly the error we want: the executable is built to run on an Android device--it does not run on Linux. Next, let's deploy it to an Android device in order to execute it.

3. Deploying the build products to the device

You can use the `adb push` command to copy build products from your Linux environment to your Android device. If you haven't already installed `adb`, you may do so via `apt-get`:

```
$ sudo apt-get install android-tools-adb
```

Once you have `adb` installed, verify your device is connected and is listed when you run the `adb devices` command - **currently this example works only in devices / emulators with at least Android 7.0, API 24** - then run the following commands to copy the Swift Android stdlib:

```
$ adb push build/Ninja-ReleaseAssert/swift-linux-
x86_64/lib/swift/android/libswiftCore.so /data/local/tmp
$ adb push build/Ninja-ReleaseAssert/swift-linux-
x86_64/lib/swift/android/libswiftGlibc.so /data/local/tmp
$ adb push build/Ninja-ReleaseAssert/swift-linux-
x86_64/lib/swift/android/libswiftSwiftOnoneSupport.so /data/local/tmp
$ adb push build/Ninja-ReleaseAssert/swift-linux-
x86_64/lib/swift/android/libswiftRemoteMirror.so /data/local/tmp
```

In addition, you'll also need to copy the Android NDK's libc++:

```
$ adb push /path/to/android-ndk-r23b/sources/cxx-stl/llvm-libc++/libs/armeabi-
v7a/libc++_shared.so /data/local/tmp
```

Finally, you'll need to copy the `hello` executable you built in the previous step:

```
$ adb push hello /data/local/tmp
```

4. Running "Hello, world" on your Android device

You can use the `adb shell` command to execute the `hello` executable on the Android device:

```
$ adb shell LD_LIBRARY_PATH=/data/local/tmp /data/local/tmp/hello
```

You should see the following output:

```
Hello, Android
```

Congratulations! You've just run your first Swift program on Android.

Running the Swift test suite hosted on an Android device

When running the test suite, build products are automatically pushed to your device. As in part three, you'll need to connect your Android device via USB:

1. Connect your Android device to your computer via USB. Ensure that remote debugging is enabled for that device by following the official instructions: https://developer.chrome.com/devtools/docs/remote-debugging.
2. Confirm the device is connected by running `adb devices`. You should see your device listed.
3. Run the tests using the build script:

```
$ utils/build-script \
  -R \                              # Build in ReleaseAssert mode.
  -T \                              # Run all tests, including on the Android device
(add --host-test to only run Android tests on the linux host).
  --android \                       # Build for Android.
  --android-ndk ~/android-ndk-r23b \  # Path to an Android NDK.
  --android-arch armv7 \            # Optionally specify Android architecture,
alternately aarch64
  --android-api-level 21
```