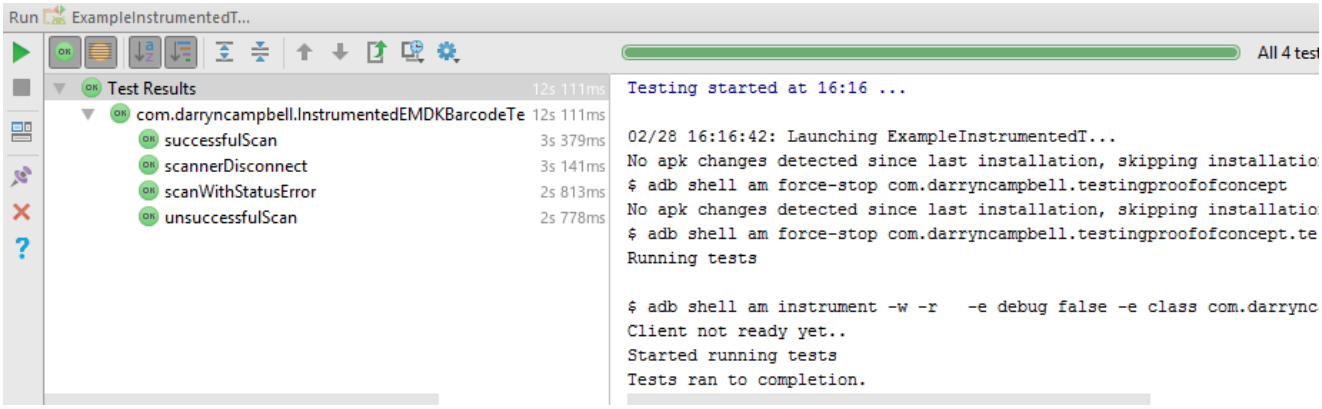
DARRYN CAMPBELL
Mobile computing and enterprise software development



Instrumented Testing and the Zebra EMDK Barcode API

1st March 2017 *10* *By* DARRYNCAMPBELL

Problem Definition

One of the big challenges developing applications for industrial mobile computers is testing. Typically, these applications will be built around data acquisition either through scanning barcodes, reading RFID tags, taking credit card payment, capturing images via the camera or capturing form data programmatically to extract text with OCR.

Any data capture will be challenging to test since it depends on manual interaction with the hardware. A barcode will be scanned or RFID tags read by pulling the device trigger, credit cards will need to be swiped with a reader generally connected via Bluetooth (with all the connectivity issues that further challenge a tester) and images or forms captured with the camera depend on user interaction to focus the camera and ensure the subject of the photo is in the field of view.

Consumer applications often rely on device emulators to expedite, automate and streamline their testing. Emulators can mock data from the orientation sensors, camera, GPS location etc. but since no emulators exist for industrial mobile devices and their data capture hardware, only a small portion of applications targeting those can be tested before moving to a real device. The move to the real device will almost always necessitate a corresponding move towards manual testing given the amount of physical interaction with the hardware required.

A Better Approach

Developers of consumer applications have for the longest time built automated testing into their application from day 1. This blog makes no attempt to give an in-depth explanation of testing frameworks since there are innumerable resources online for that and if you are looking for information on testing Android applications a good place to start is Google's own documentation at https://developer.android.com/training/testing/index.html#start.

Testing takes two forms:

1. Local tests for individual algorithms and classes, these run within the local JVM and therefore do not have access to Android APIs.
2. Instrumented tests which comprise of both unit tests and more complex user interface tests. Instrumented tests run on the device.

On Android, it is recommended you would write your tests using JUnit and extend the capabilities of your tests to handle UI interaction using the Espresso framework. Again, the Android documentation goes into a lot more detail here and is essential reading for anybody unfamiliar with the topic.

THE GOAL IS TO CREATE AUTOMATED JUNIT TESTS WHICH EXERCISE THE DATA CAPTURE HARDWARE.

Mocking the Hardware

In many industries, the solution to test code modules that depend on external hardware is to mock that hardware, or at least the interface to it. As an application developer, to test data capture hardware in an automated fashion you would either want a full emulator or at least be able to mimic the interface to that hardware in your instrumented tests.

Unfortunately, as previously mentioned **no full emulators are available for industrial mobility devices** and **integration with test frameworks is not offered** out of the box.

Options for Capturing Data

Concentrating on Zebra Android mobile computers such as the TC51 and TC75 there are a number of ways to interact with the data capture hardware:

- Through a dedicated Android API
- Through a dedicated Xamarin API
- Through a dedicated JavaScript API
- Through Intents (DataWedge)
- Through some combination of the above e.g. Android API + DataWedge.

And each device has dedicated hardware for capturing data:

- Barcode scanner
  - SimulScan to capture form data and perform OCR
- RFID reader (on selected models)
- NFC reader (on selected models)
- Bluetooth to connect with:
  - Payment devices
  - Barcode scanners
  - Printers
  - Etc.

- Card reader (on selected models)
- USB connected hardware

Clearly a single testing solution to cover all scenarios is unrealistic and Zebra do not offer any test variants of their device interfaces. It is therefore left as an exercise to the developer to mock the hardware interface as required.

One Possible Solution

As previously mentioned, the number of combinations of API languages and data capture hardware is large. I therefore present a single solution to the most common use case, **testing barcode scanning through the Android API**. Obviously, it would be possible to mock other APIs but please take this as a proof of concept.

The EMDK Barcode API

The EMDK Barcode API is the API used to interface with the barcode scanning hardware on Zebra mobile computers and is documented here:

The main class (com.symbol.emdk.barcode) returns:

- Status to the application via the Scanner.**StatusListener**
- Data (decoded barcodes) to the application via the Scanner.**DataListener**
- Scanner connectivity changes to the application via the BarcodeManager.**ScannerConnectionListener**. This is used primarily for Bluetooth connected scanners.

In order to test an application that utilizes the barcode scanner we need to be able to invoke these interfaces with test data whilst running our instrumented tests, for example.:

```
// Click the start scan button
onView(withId(R.id.buttonStartScan)).perform(click());

// Simulate a barcode being scanned
mockedInterface.AddScanData("123456789");
ScanDataCollection scanDataCollection = mockedInterface.ReportScan(success);

// Trigger the data listener
activity.onData(scanDataCollection);

// Test that the correct data was scanned
onView(withId(R.id.textViewData)).check(matches(withText("0123456789\n")));

// Click the stop scan button
onView(withId(R.id.buttonStopScan)).perform(click());
```

It feels as though the Barcode interface was designed specifically to make it difficult to mock the returned data, though I am sure this is not the case(!) The data types returned by the various interface methods only have private constructors, for example the payload of the onStatus method is a StatusData object. In order to mock the onStatus call we need to create a StatusData object but no public constructors exist (even the default constructor) and there are no setters exposed on the public interface.

We are therefore required to use Java reflection to create these interface payload objects. Obviously since this technique is not officially supported by Zebra there is a possibility they will change their internal data structures or refactor the private code at some point but for the purposes of this blog, all samples have been written to work with EMDK 6.0, the latest release at the time.

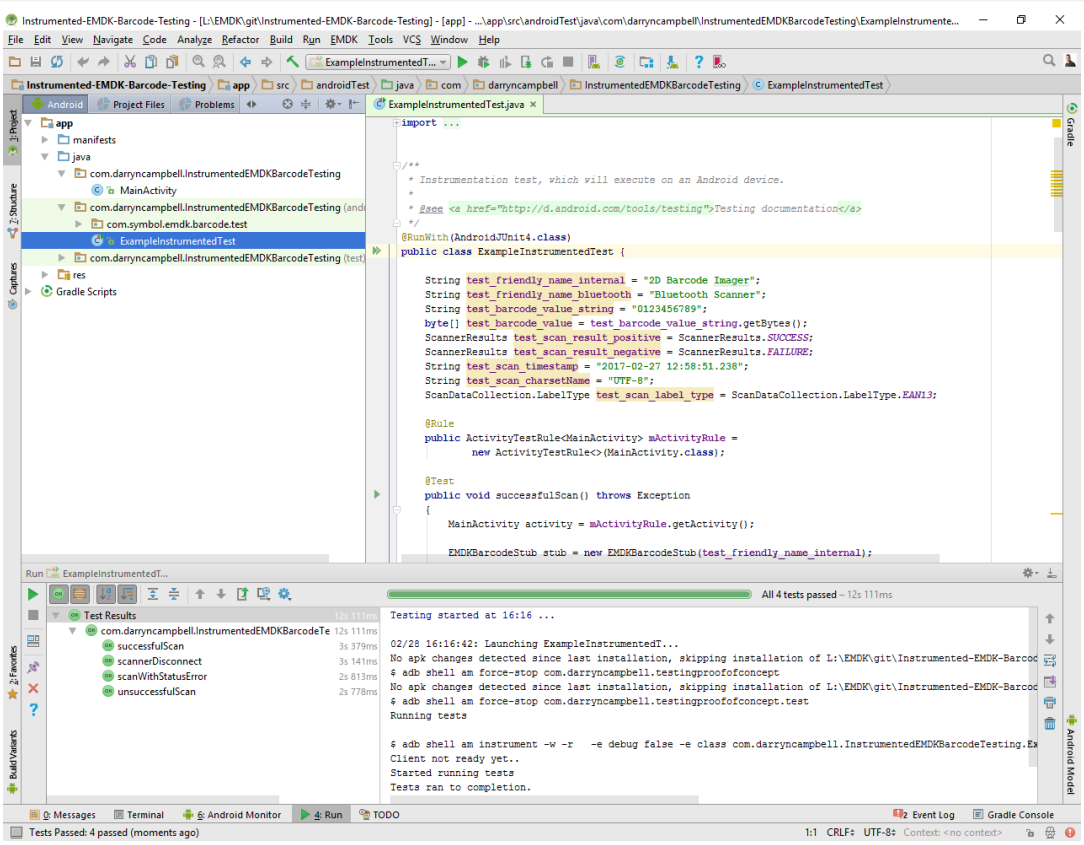
I have done the hard work of stubbing the listener payload objects in a helper class here: [Github Link]. There are a number of methods:

- ReportStatus
  - Returns a StatusData object which can be passed to the StatusListener. Takes the desired status as a parameter.
- AddScanData
  - A single barcode scan (trigger press) can contain multiple decoded barcodes, for this reason mocking a scan has two phases and in the first you call this method multiple times to add all the data you want the scan to return.
  - Each call to AddScanData takes the barcode data you want returned, the symbology of the read barcode and a timestamp.
- ReportScan
  - Once all the barcode data has been given through AddScanData, call this method to return a ScanDataCollection object with the mocked data. ScanDataCollection is the payload passed to the DataListener. You can also choose to have the scanner report that the read failed through this API.
- CreateScannerInfo
  - Creates a mocked ScannerInfo object using the passed attributes to describe the scanner. Defining a scanner takes a lot of attributes so you may wish to refer to the example at [Github Link].
  - The onConnectionChange interface method takes a ScannerInfo object along with a BarcodeManager.ConnectionState object to indicate whether that scanner is connected or disconnected.

Worked Example

I have put together a very simple example of how to test the barcode scanner with mocked data:

- The example is available on GitHub, https://github.com/darryncampbell/Instrumented-EMDK-Barcode-Testing.
  - Clone the repository and load into Android Studio.
  - Take care to modify the app build.gradle file to point to your EMDK installation within your Android add-ons directory.
- You must run the sample on **a Zebra mobile computer**, you cannot run on an emulator as the mocked interface depends on libraries available on the device.
  - If you have an older JellyBean device you may need to update the device runtime, see the EMDK release notes.
- The tests are built around Zebra's ubiquitous BarcodeScanner1 sample project
- Four tests are defined in the instrumented test file.
  - Testing a successful scan
  - Testing an unsuccessful scan
  - Testing reception of a status error whilst scanning
  - Testing a Bluetooth scanner disconnecting.
- All tests make use of a helper class which encapsulates the logic of reflecting the EMDK interface to create objects which can be returned through the listener interfaces, as described in the previous section



The proof of concept tests running on a TC55 device, mocking the scanner hardware.

Share this:

Related

Deploying an application to Zebra Android devices ranging from Jellybean to Marshmallow and beyond
16th January 2017
In "Zebra Technologies"

Writing Enterprise Android applications that capture barcode data and run on multiple devices
16th August 2016
In "Zebra Technologies"

DevTalk: DataWedge v6.3 - Benefits and Challenges
16th August 2017
In "Speaking"

Category *Zebra Technologies*

Tags *Android* *EMDK* *Testing*

10 Comments

Este says:

2nd April 2017 at 12:54 pm

Thanks Darryn; this is a really useful post. I would have balled my eyes having blindly hit some of the obstacles you've outlined – e.g. no out of the box test framework support / having to use reflection. I will give this a go with the MC18 and I'll let you know how I get on.

Reply

 ***Darryn Campbell*** says:

2nd April 2017 at 7:57 pm

Thanks, yes please let me know. Right now this kind of feature (automated testing) is not a priority for the development team but that would change if more people were asking for it.

Reply

 ***Ernesto Gonzalez*** says:

20th June 2017 at 1:37 pm

Thank you Darryn, interesting information and I would ask you about the same test for Xamarin Android. I'm working with TC55 and RFD8500. Obviously I cannot test TC55 with the emulator but right now we do not have the physical device. My guess is about create a helper class (Android Callable Wrapper) for the EMDKManager.IEMDKListener interface since the app has 5 modules and each are using barcode. I'll appreciate any comment or proposal. Thanks.

Reply

 ***Darryn Campbell*** says:

21st June 2017 at 9:03 am

Hi Ernesto, so for the Xamarin component, Zebra just wrap the Android Jar file from the EMDK so one possibility would be to create your own binding (the process is explained here https://developer.zebra.com/docs/DOC-2813) but full disclosure, I could never get that to work for myself back in 2015 when the EMDK Jar was much simpler so that approach might be non-trivial!

The other difficulty you will have is the RFD8500 SDK does not support Xamarin (see this thread from last year for more info: https://developer.zebra.com/message/89514#89514). Looking at the SDK, it seems nothing has changed in the past year.

Beyond that I'm afraid my Xamarin experience is limited, I extensively used reflection in the above post and sample app to mimic the data types being returned by the scanner but I am uncertain if that would work in Xamarin. There is a reflection namespace (https://developer.xamarin.com/api/namespace/Java.Lang.Reflect/) but there are not too many posts / pages I can see on Google to help with that. I had to use reflection to create some of the types e.g. ScanDataCollection – if you can create those with Xamarin without reflection then it might be simpler to use a wrapper class: http://techdocs.zebra.com/emdk-for-xamarin/2-4/api/barcode/ScanDataCollection/

Hope that helps.

Reply

Bashar Asaad says:

15th February 2018 at 6:51 am

Thank you Darryn, interesting information and I would ask you about the Android Studio configuration because i tried to run your example on my android studio but i get the same error when i try to run the test :
java.lang.NoClassDefFoundError:

com.darryncampbell.InstrumentedEMDKBarcodeTesting.MainActivity
at com.darryncampbell.InstrumentedEMDKBarcodeTesting.ExampleInstrumentedTest.
(ExampleInstrumentedTest.java:36)
at java.lang.reflect.Constructor.constructNative(Native Method)
at java.lang.reflect.Constructor.newInstance(Constructor.java:423)
at org.junit.runners.BlockJUnit4ClassRunner.createTest(BlockJUnit4ClassRunner.java:217)
at org.junit.runners.BlockJUnit4ClassRunner$1.runReflectiveCall(BlockJUnit4ClassRunner.java:266)
at org.junit.internal.runners.model.ReflectiveCallable.run(ReflectiveCallable.java:12)
at org.junit.runners.BlockJUnit4ClassRunner.methodBlock(BlockJUnit4ClassRunner.java:263)
at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:78)
at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:57)
at org.junit.runners.ParentRunner$3.run(ParentRunner.java:290)
at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:71)
at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:288)
at org.junit.runners.ParentRunner.access$000(ParentRunner.java:58)
at org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:268)
at org.junit.runners.ParentRunner.run(ParentRunner.java:363)
at org.junit.runners.Suite.runChild(Suite.java:128)
at org.junit.runners.Suite.runChild(Suite.java:27)
at org.junit.runners.ParentRunner$3.run(ParentRunner.java:290)
at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:71)
at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:288)
at org.junit.runners.ParentRunner.access$000(ParentRunner.java:58)
at org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:268)
at org.junit.runners.ParentRunner.run(ParentRunner.java:363)
at org.junit.runner.JUnitCore.run(JUnitCore.java:137)
at org.junit.runner.JUnitCore.run(JUnitCore.java:115)
at android.support.test.internal.runner.TestExecutor.execute(TestExecutor.java:58)
at android.support.test.runner.AndroidJUnitRunner.onStart(AndroidJUnitRunner.java:375)
at android.app.Instrumentation$InstrumentationThread.run(Instrumentation.java:1701)

Reply

darryncampbell says:

22nd February 2018 at 11:03 am

I couldn't tell you when it stopped working but it looks like at some point gradle started requiring you to specify "provided" files for tests separately. I have fixed this at https://github.com/darryncampbell/Instrumented-EMDK-Barcode-Testing/commit/1870906237f36540936c392949a8ca553879d2d3.

Thank you for raising.

Reply

TJ says:

9th July 2018 at 7:01 am

Hi Darryn,

Thanks for your handwork. it is truly appreciated.

trying to run successfulScan() of ExampleInstrumentedTest.java but endup with below error. can you suggest how to proceed,

Installation failed with message INSTALL_FAILED_MISSING_SHARED_LIBRARY: Package couldn't be installed in /data/app/com.darryncampbell.testingproofofconcept-1: Package com.darryncampbell.testingproofofconcept requires unavailable shared library com.symbol.emdk; failing!.

Reply

darryncampbell says:

9th July 2018 at 7:48 am

Hi, couple of things to check:
– You have EMDK for Android installed (you should see add-ons for symbol_emdk under your Android add-ons folder, e.g. mine is C:\Users\darry\AppData\Local\Android\Sdk\add-ons on Windows
– You are running on a Zebra device which supports EMDK

Reply

Serge says:

18th October 2019 at 4:51 pm

Hi Darryn,

Thank you for your very helpful post.
You mentionned "Unfortunately, as previously mentioned no full emulators are available for industrial mobility devices and integration with test frameworks is not offered out of the box.".
Do you have any lead to be able to run it on an emulator? Your post having 2.5years now, hoping there were some upgrade ^^'
The main issue being the EMDKManager trying to get info about device scanner when obviously, emulator doesn't have one.
Thanks in advance for your help

Reply

darryncampbell says:

18th October 2019 at 7:11 pm

Hi Serge, I have since published https://developer.zebra.com/blog/test-your-zebra-scanning-application-emulator on the official developer portal. To summarize, although there is no Zebra emulator available it IS possible to test applications that use DataWedge to interface with the scanner on an emulator. Further, DataWedge is the recommended approach for adding scanning to your application, see the big red box at https://techdocs.zebra.com/emdk-for-android/7-4/guide/about/.

Regarding your specific question, there are still no plans to release an official emulator image for Zebra devices and no plans to support EMDK on any emulator.

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

Notify me of follow-up comments by email.

Notify me of new posts by email.

Post Comment

SEARCH