# Ezone Indoor Navigation Frontend Test Cases

Lei Wang
email: 21676963@student.uwa.edu.au

September 9, 2017

## 1 Appium for both iOS and Android Test

Table 1 lists top 5 testing framework for your daily Android builds, creation, and correction.

Table 1: Top 5 testing frameworks.

| | Robotium | uiautomator | Espresso | Appium | Calabash |
|---|---|---|---|---|---|
| Android | Yes | Yes | Yes | Yes | Yes |
| iOS | **No** | **No** | **No** | Yes | Yes |
| Mobile web | Yes(Android) | Limited to x.y clicks | No | Yes (Android and iOS) | Yes (Android) |
| Scripting Language | Java | Java | Java | Almost any | Ruby |
| Testing creation tools | Testdroid Recorder | UI Automator viewer | Hierarchy Viewer | Appium.app | CLI |
| Supported API levels | All | **16 or higher** | **8, 10, 15 or higher** | All | All |
| Community | Contributors | Google | Google | Active | **Pretty quiet** |

Appium is an open source, cross-platform test automation tool for native, hybrid and mobile web and desktop apps, tested on simulators(iOS), emulators(Android), and real devices (iOS, Android, Windows, Mac). There are several reasons to choose appium as a test automation tool for our Ezone Indoor Navigation applications:

- You don't have to recompile your app or modify it in any way, due to use of standard automation APIs on all platforms.
- You can write tests with your favourite dev tools using any WebDriver-compatible language such as Java, Objective-C, JavaScript with Node.js (in promise, callback or generator flavors), PHP, Python, Ruby, C#, Clojure, or Perl with the Selenium WebDriver API and language-specific client libraries.
- You can use any testing framework.

Based on above analysis, Appium is more promising for our Ezone project.

## 2 UI Test

In this test, we do visualization test for buttons, floor plans, shortest path, etc.

## 2.1 Buttons

- Check whether the button is in the correct location;
- Check the size of the button whether it is appropriate or not;
- Check the link of each button, once click should give correct response.

## 2.2 Zoom in & zoom out

- Check the floor plan whether it can be zoomed in and out properly;
- Check once switch to a different floor plan, it can still be zoomed in and out properly.

## 2.3 Some Key Operations

- Check if the user can put a marker on the selected location;
- Check once the marker is shown on the floor plan, a pop-up window which shows room details will appear correctly;
- Check if the pop-up window shows the related room information correctly;
- Check if the pop-up window has a button so that the user can use this button to **start tracking**.
- Check if the user press a location which is out of the building area, the application should give a message to remind the user that it is out of the navigation area.

## 2.4 Shortest Path

- Check if there is a visible path when the user inputs the destination;
- Check whether the path seems reasonable or not (the path should not go through any walls);
- Check whether the given path is optimal (the shortest path) or not.

## 2.5 Navigation

- Check if it can show user's current location correctly (try different floors whether it can detect the floor changes), whether the error is within a reasonable area (The radius of the error should be no more than 3 meters);
- Check if this application can do single floor navigation correctly;
- Check if this application can do multiple floor navigation correctly;
- Check if the path updated correctly when the user moves in the building;
- Check if there is a door closed (a path not available), it can give another path or messages ('There is no way to go to XXX.') when necessary.
- Check if the user reaches the destination, the application should give a message to remind the user such as 'You are arrived!'.

## 2.6 Other tests

- Check whether the keyboard can appear and disappear when necessary;
- Check the welcome screen when the application opens;
- Check the response time when interacting with this application (try it in a different environment such as weaker signal area, etc.);
- Check whether this application can work for different versions of platform.

# 3 Unit Test

## 3.1 Code Inspection with Lint

In addition to testing your Android application to ensure that it meets its functional requirements, it's important to ensure that your code has no structural problems. Poorly structured code can impact the reliability and efficiency of your Android apps and make your code harder to maintain. For example, if your XML resource files contain unused namespaces, this takes up space and incurs unnecessary processing. Other structural issues, such as use of deprecated elements or API calls that are not supported by the target API versions, might lead to code failing to run correctly.

The lint tool checks your Android project source files for potential bugs and optimization improvements for correctness, security, performance, usability, accessibility, and internationalization. When using Android Studio, configured lint and IDE inspections run whenever you build your app. Figure 1 shows how the lint tool processed the application source files.

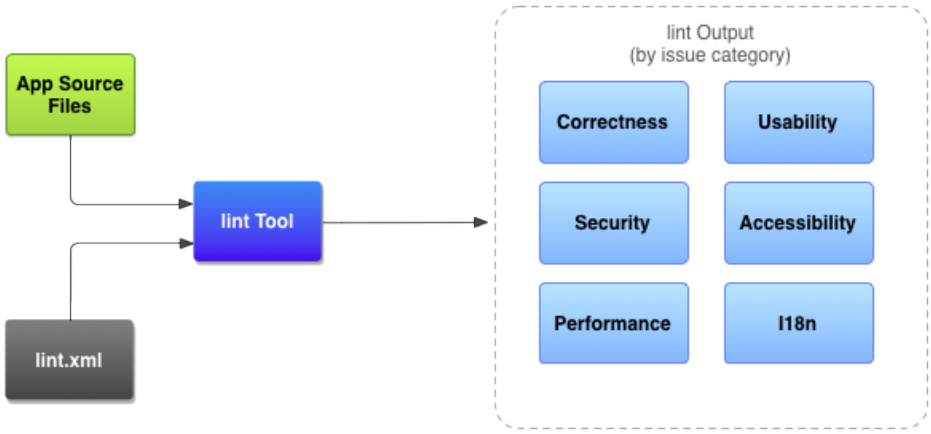


Figure 1: Code scanning workflow with the lint tool.

The detailed operation in Android Studio can be found in:
https://developer.android.com/studio/write/lint.html#manuallyRunInspections

## 3.2 Code Coverage in Xcode 7

Code coverage is a feature in Xcode 7 that enables you to visualize and measure how much of your code is being exercised by tests. With code coverage, you can determine whether your tests are doing the job you intended.

Code coverage in Xcode is a testing option supported by LLVM. When you enable code coverage, LLVM instruments the code to gather coverage data based on the frequency that methods and functions are called. The code coverage option can collect data to report on tests of correctness and of performance, whether unit tests or UI tests.

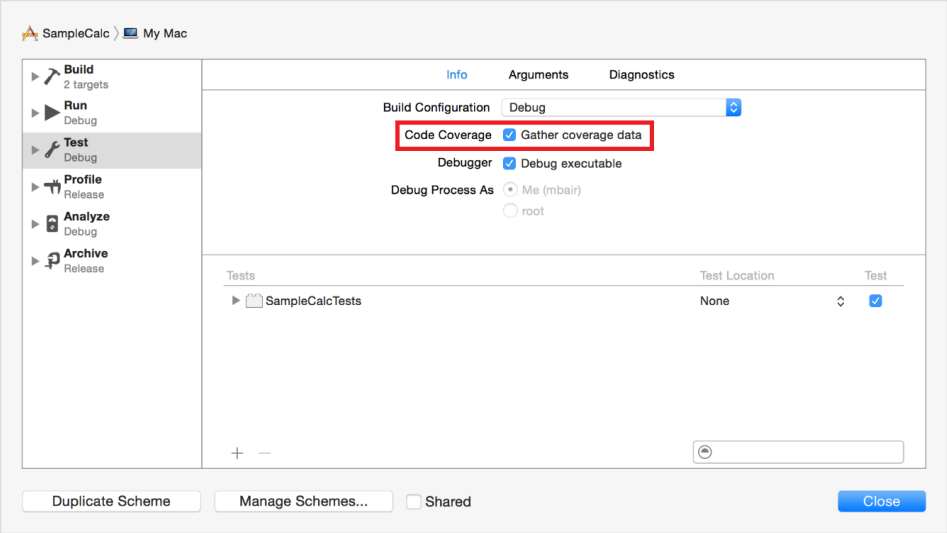


Figure 2: Code coverage operation in Xcode.

The detailed operation for code coverage test in Xcode can be found in:
https://developer.apple.com/library/content/documentation/DeveloperTools/Conceptual/testing_with_xcode/chapters/07-code_coverage.html

## 3.3 Local unit tests & Instrumented tests

Android Studio is designed to make testing simple. With just a few clicks, you can set up a JUnit test that runs on the local JVM or an instrumented test that runs on a device. Of course, you can also extend your test capabilities by integrating test frameworks such as Mockito to test Android API calls in your local unit tests, and Espresso or UI Automator to exercise user interaction in your instrumented tests. You can generate Espresso tests automatically using Espresso Test Recorder.

The detailed operation in Android Studio can be found in:
https://developer.android.com/studio/test/index.html

# 4 Acceptance Test

The following test cases are designed for both Android and iOS platform, and Appium is used as a test automation tool.

### 4.1 Current location button

The current location button should highlight the user's position correctly. It should automatically detect which floor does the user located on. Once the button is clicked, the floor plan should be updated automatically and correctly. In addition, the radius of the error should be within 3 meters.

### 4.2 Buttons to change floors

The floor number buttons (or some method of being able to choose floor plans) are used to select floor plans when necessary. Once the button is clicked, the floor plan should be updated accordingly based on the selection.

### 4.3 Select destination

The users are allowed to place a marker on a destination, and once the user click a location on the floor plan, a marker should show up. The marker should be placed exactly where the user clicked on the floor plan.

### 4.4 Room markers & meta-data pop-up window

The floor plan should visually display markers for rooms which can be clicked. Once the user click on the marker, a pop-up notification style window should show up which includes the room information.

The pop-up window should have two buttons, which enable the user to view more room information or start to navigate to this location.

### 4.5 Live updating of path

The visible path should be displayed on the screen. The path should be a reasonable path, which means it should be an optimal or a shortest path.

Once the user moves, the path should be updated accordingly based on the movement of the user.

### 4.6 Response time

When the user is interacting with this application, it should response quickly without letting the user waiting for too long to get the current location, room information or visible path.

## Acknowledgements

Some resources include texts and pictures are obtained from online resources.