Android Test Automation Framework [Documentation]

2011/08/03, V1 (r3)

Philip Peng <t-phpeng>

SDET Intern, Summer 2011

Devices & Roaming (DRX), Windows Live

Manager: Anup

Coach: James

Table of Contents

Overview

Purpose

This was an intern project I, Philip Peng (t-phpeng), carried out and completed during the second half of my Summer 2011 internship as an SDET Intern. The project's goal was to create a working test automation framework for the Android platform. This test framework would then be later modified to fit in with a common test automation framework being developed for the three mobile platforms targeted by the Mobile SkyDrive team: iOS, Windows Phone 7, and Android. The Mobile SkyDrive team is part of Windows and Windows Live (WWL)'s Devices & Roaming Team (DRX) and is working on the (first version of the) SkyDrive mobile client app.

There currently exist two possible alternatives for this framework. The first is the open-source "Robotium" framework, which already fully-supports blackbox scenario testing and is very stable. Why not use Robotium then, which has already been thoroughly tested? Doing so would have easily completed the command-executing part of this project; however, usage would also require permission from Microsoft's legal departments. I was told that, given a choice between using pre-existing open source tools and developing an internal tool that does the same thing, I should do the latter. And so I did (although for some parts, I did take a look at the Robotium code for ideas on implementation method).

The second alternative would be to use the Bing team's existing Android test automation framework, which supports both the Android and RIM platforms. This framework has also been in use for quite some time and thus is known to work. Usage of this framework would also have completed most of the command-executing part of the project as well as add RIM support; however, usage would require heavy modification. The framework ran tests by reading in full test cases encoded in XML stored on the Android device's SD card, thus making the list of test commands something static that must be pre-prepared, then log the results back to the SD card for inspection later. For our test framework, we needed something that would receive commands over the network and respond immediately in a dynamic manner. In addition, dialogs were not supported due to the framework only supporting one level of UI elements (as opposed to the many that will appear in our app). Considering the multiple layers of abstraction used and consequently the extremely large codebase, modifying the Bing team's framework would be extremely time-consuming and, by my estimates, take longer than the time it would take to rewrite an entirely new one (given that I only had ~5 weeks of time to complete the project).

Ultimately, this Android Test Automation Framework fulfills three important requirements that were lacking in the other two test frameworks: 1) dynamic execution of commands received over the network, 2) integration readiness with the Mobile SkyDrive team's existing common test framework, and 3) availability of a complete set of scripts for managing the rest of the device-related test automation process. In some aspects this framework has some advantages over the other two but lacks maturity and thorough testing (due to lack of time). Over time, however, I do believe it can be further developed to encompass all the features of Bing team's framework and Robotium, but be more powerful due to being part of a common test framework that supports iOS, Windows Phone 7, and Android.

Components

The Android Test Automation Framework contains four main components: *1)* PC Client, *2)* PC Scripts, *3)* Android Test Runner, and *4)* Android Test Application. The Host PC runs *1)* and *2)*, and the Android device runs *3)* and *4)*. Note that the PC Scripts must be run on the Host PC that the Android device is physically connected to (via USB) whereas the PC Client can be run on any machine on the same network as the Host PC (assuming it supports HTTP requests).

The *1)* PC Client (TestClient) is a PC program that sends command requests (e.g. test case commands) to the Android device running the Android Test Runner. The *2)* PC Scripts (Scripts/*.bat) are a series of scripts that handle Android device management (e.g. package updating and launching). The *3)* Android Test Runner (MobileSkyDriveTest) is a custom Android instrumentation test case, run by the Android OS's built-in InstrumentationTestRunner application, which extends testing capability by implementing an HTTP server that dynamically receives commands and executes them. The *4)* Android Test Application (MobileSkyDrive) is the production Android application that is under instrumentation and testing.

Status

At the time of writing this document, there does not yet exist[1] a MobileSkyDrive app for Android; the temporary MobileSkyDrive app is actually a prototype that only implements authentication, simple browsing, and a basic UI. As well, the common test automation framework sends commands as SOAP messages; however, there is no built-in SOAP deserialization code generator for

[1] Planned for M3

Android's Java implementation. Due to lack of time, a temporary, custom message specification also using the XML format was used, with MobileSkyDriveTest expecting to receive messages in that custom specification and the temporary TestClient used to construct and send those custom messages.

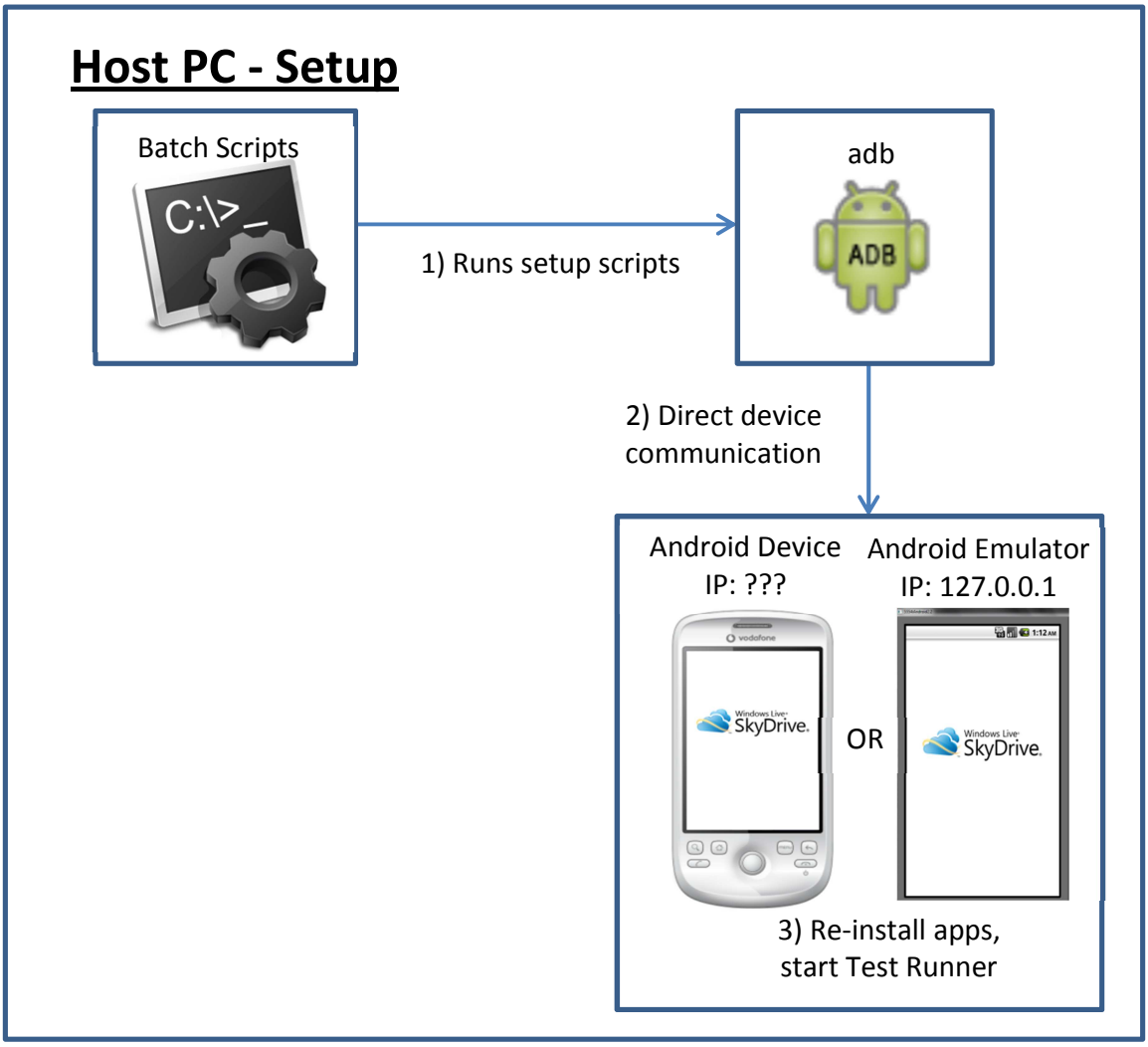
Apart from that, the current revision of the Android test framework is usable right now as a standalone test framework. It will be ready for integration with the common test framework once the SOAP deserializer is implemented and can be used immediately to test any future MobileSkyDrive app or any other Microsoft-developed app[2].

[2] Only if source code access is available as of the moment – see the To-Do note on dynamic class loading.

Process

Setup

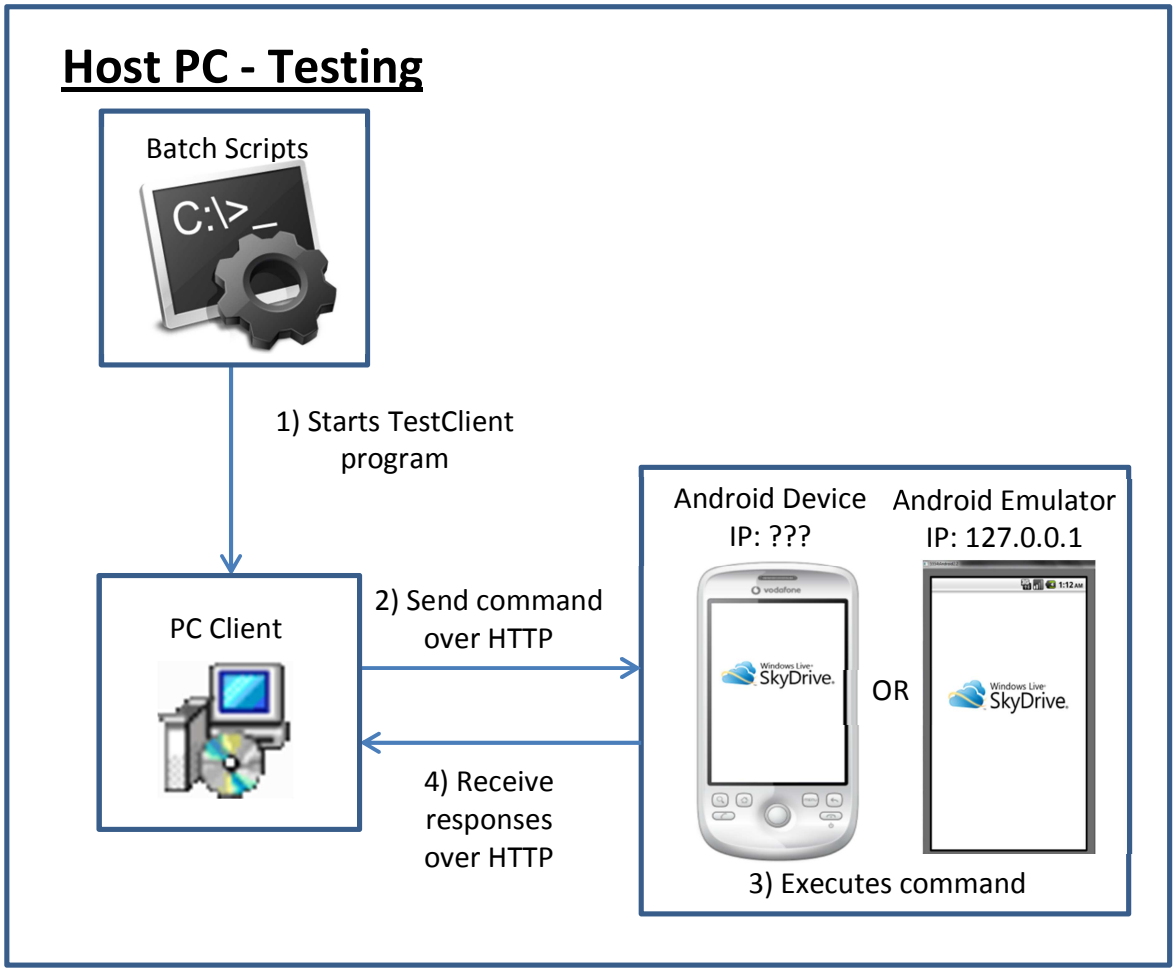
To start off the process, the Android device needs to first install the latest versions of the Test Runner and Test Application and then launch the Test Runner. This is done by executing a script on the Host PC, which will communicate with the Android device via the Android Debug Bridge (adb) command line tool. Note that it is necessary to use the adb tool for installation and uninstallation of the app as opposed to remote wireless deployment due to security restrictions[3].



[3] See the To-Do note on spoofing GTalkService

Testing

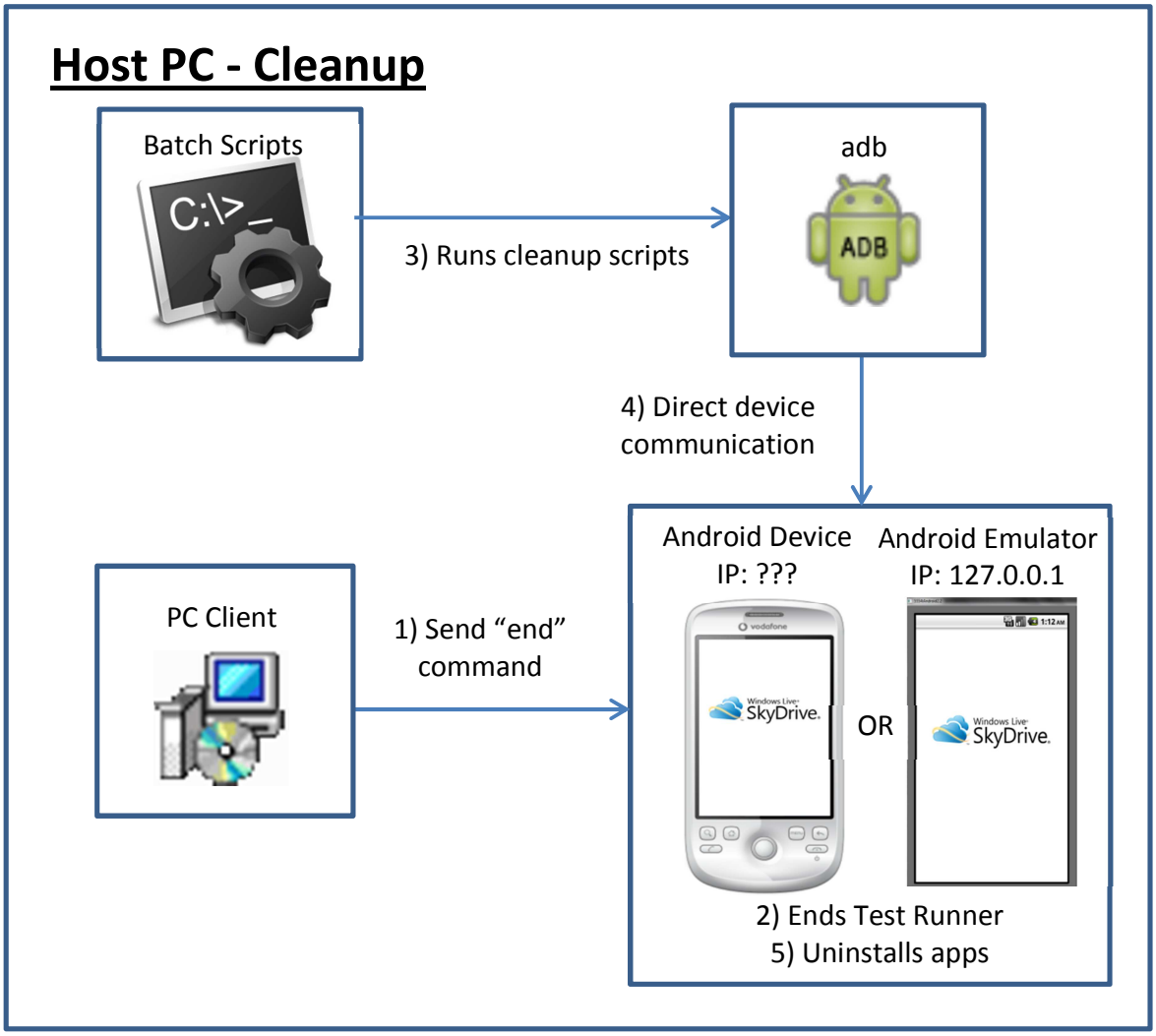
Once the Test Runner has launched, in the case of an Android emulator, the Host PC can communicate with the Android emulator at the localhost address (127.0.0.1) once port forwarding has been set up. In the case of a physical Android device, the Host PC can communicate with the Android device if the device's IP address is recorded[4]. This communication is done via the interactive PC client (TestClient) program, which sends commands and receives responses over HTTP.



[4] There is currently no script for obtaining the IP address of connected physical devices – see the To-Do note on retrieving IP address

Cleanup

After the testing is complete, an "end" command is sent by the PC Client (TestClient), which tells the Android Test Runner to finish the test (e.g. exit the Test App, close the server, and end the Test Runner). Another script will then tell adb to uninstall the Test Runner and Test App on the Android device or emulator. If an emulator is used, another script will also tell the Host PC to kill the emulator process[5].

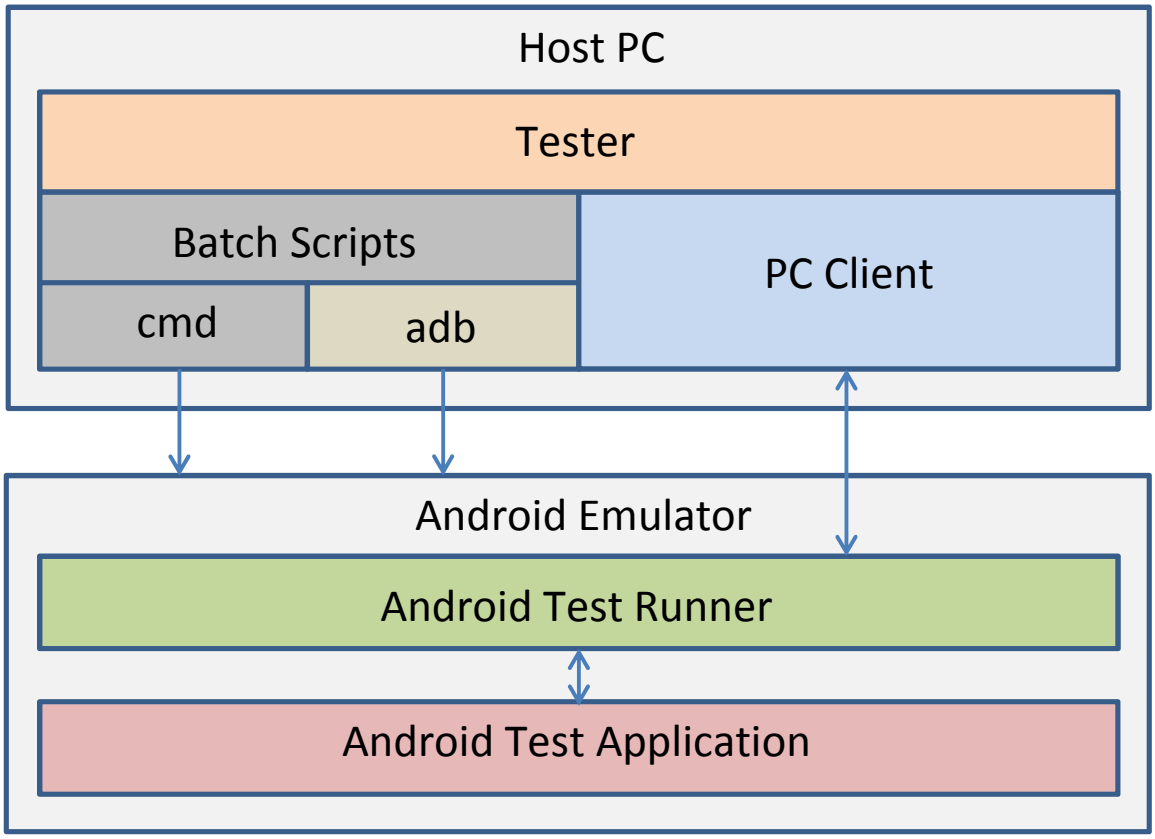


[5] There is no graceful way of quitting of the emulator program other than clicking the "X" button or telling the OS to kill the process

Abstraction Layers

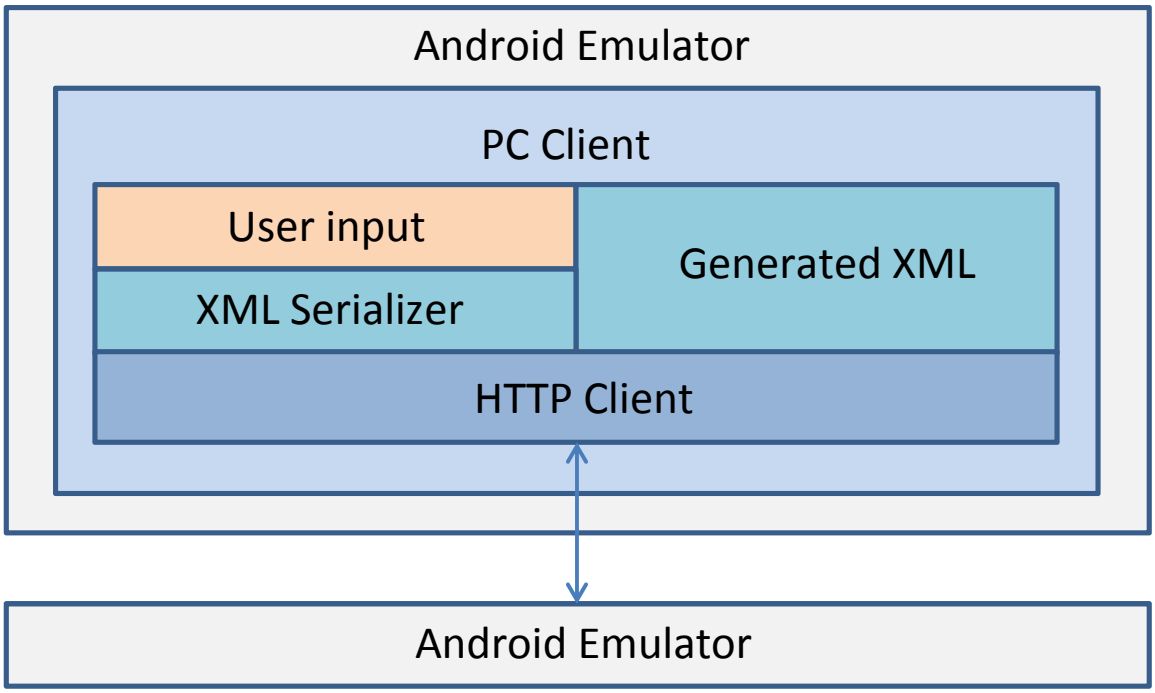
PC-to-Android

Communication between the PC and the Android device is done via the Android Debug Bridge (adb) program and via the PC Client. The adb program communicates with the Android device via installed drivers, and the PC Client communicates with the Android Test Runner via XML-encoded messages sent via the HTTP protocol (see next section).



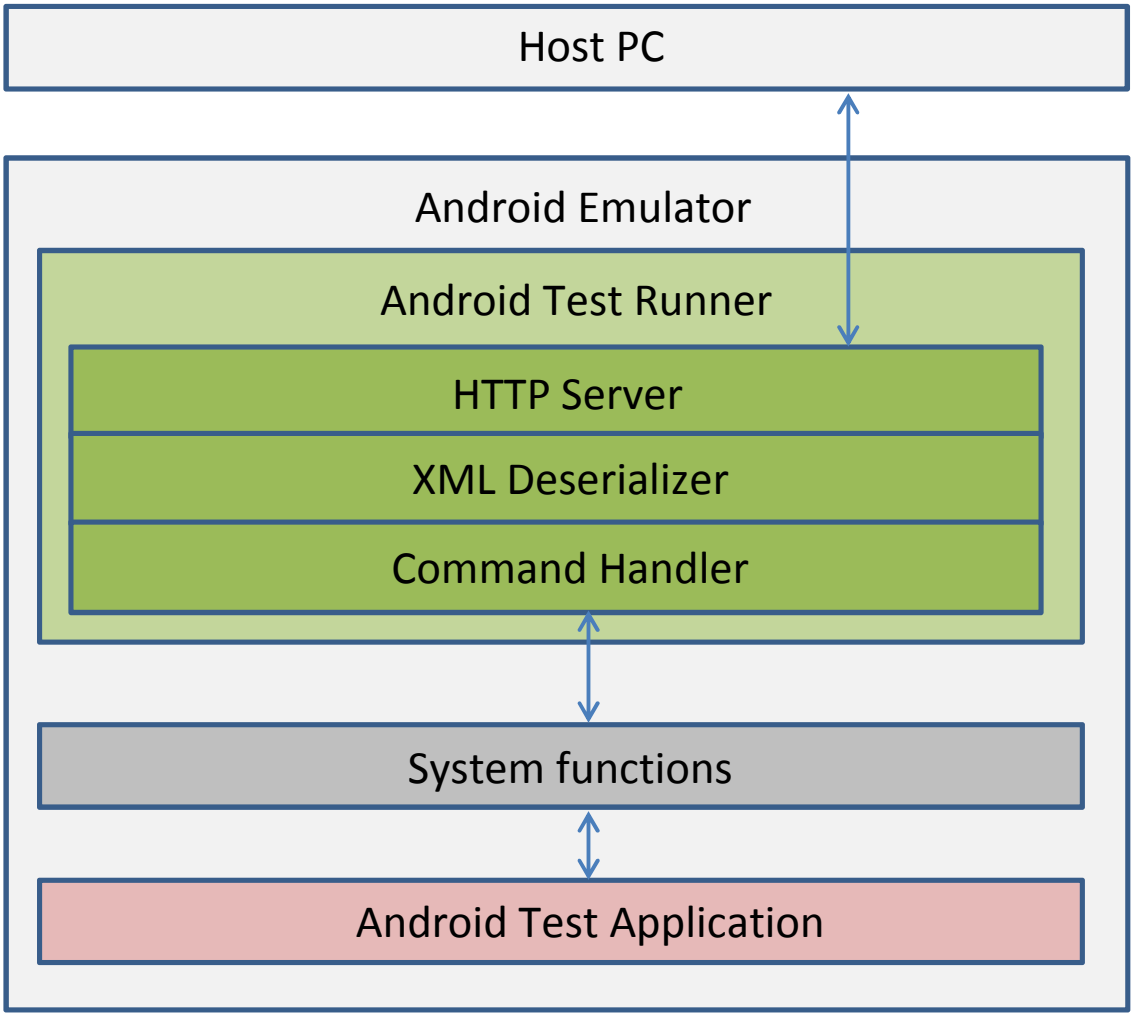
PC Client

In the PC Client itself, input is read from the user (via interactive command line prompts) or from a file (pre-generated XML file). For user input, a command is built and serialized into XML format. This XML message is then sent to the Android Test Runner over HTTP protocol, with the PC Client acting as an HTTP client. In the future, the PC Client used here (TestClient) will be replaced by the PC Client used in the common test framework and commands will be sent via SOAP messages.



Android Test Runner

In the Android Test Runner, an HTTP server is actually set up and running during the duration of the test. The server listens to port 8080, a hardcoded value that can be changed to anything above 1023 (as ports 0-1023 require root access as per Linux defaults). When an HTTP request is received, the HTTP entity (body) is parsed through an XML deserializer[6] to reconstruct the requested command. This deserialized command is then handled by a Command Handler which will then execute the corresponding test command on the Android Test Application (via Android system calls or code injection).



[6] The server code in MobileSkyDriveTest currently assume that all HTTP requests are in XML format