

2016

User Guide for Cafe Mobile With SeeTest Framework

This document gives a basic guideline for using CAFÉ Mobile framework With Seetest tool, different features provided in it. It also details out creating and running first automation scripts, using CAFÉ Mobile with Seetest



Table of Contents

Summary:	3
Skill set required for using Café Mobile with SeeTest:	3
Tool set required for Café Mobile Test Automation:.....	3
Setting Up Selenium with Café Mobile framework:	3
Structure of the Project:	5
Mobile Test Automation Configuration	12
Mobile NativeApp Test Execution on SeeTest Cloud Device:	13

Summary:

Café Mobile with SeeTest Automation supports Execution for Functional, Cross browser and Cross platform testing over Mobile Browsers, Native App testing over handheld devices, Emulators and Cloud Devices. This version provides integration between Selenium WebDriver and SeeTest.

All the variants provide integration with CI tools like Jenkins.

Skill set required for using Café Mobile with SeeTest:

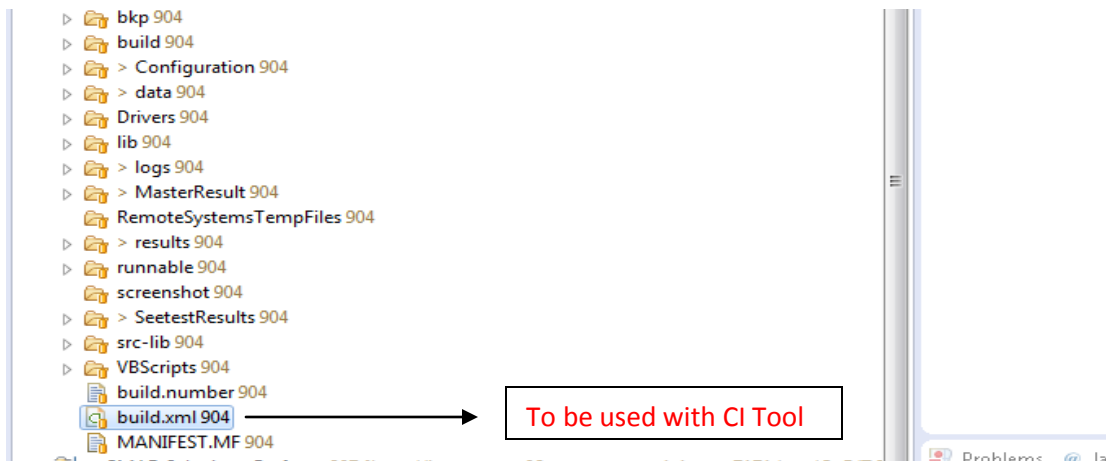
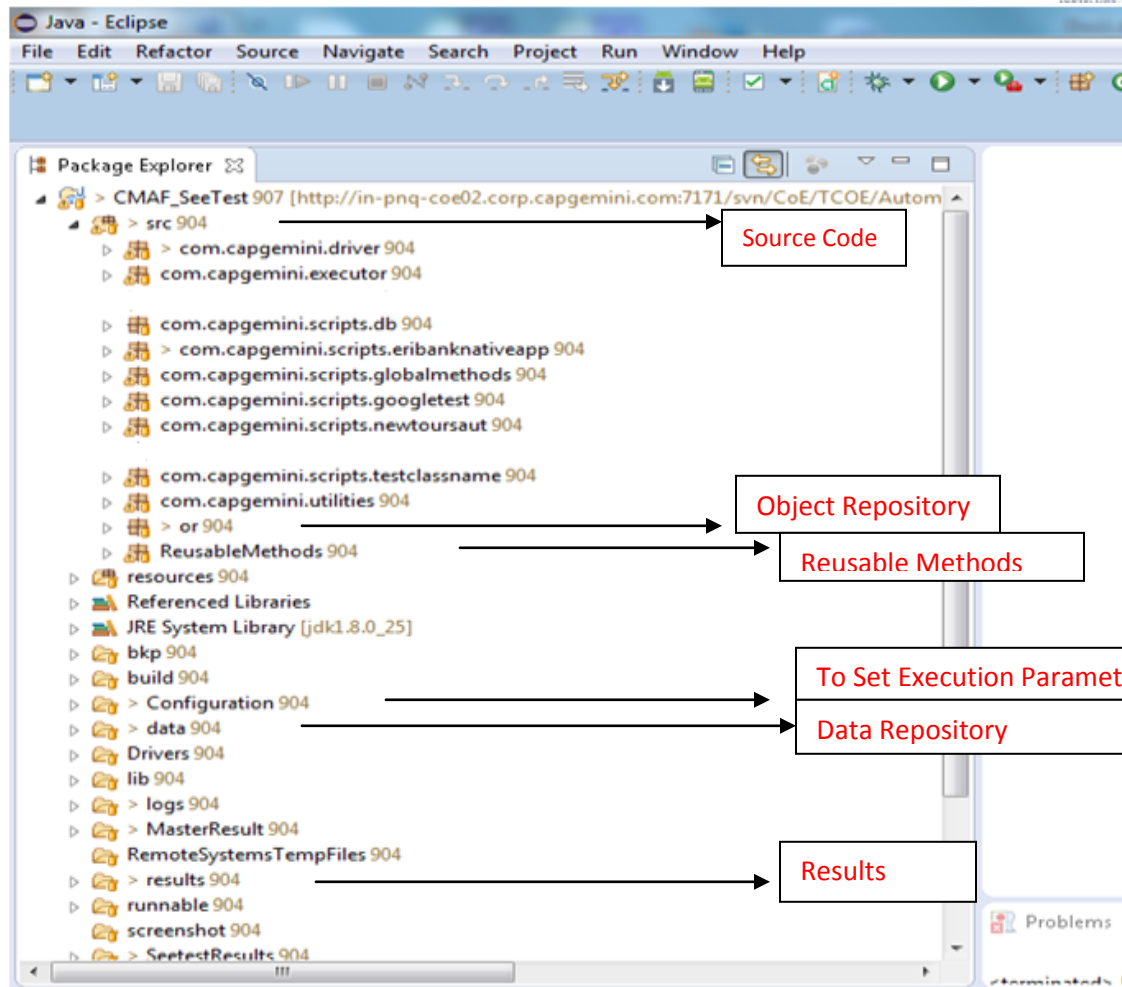
- 1) Basics of Automation Testing
- 2) Basics of Java
- 3) Basics of Selenium WebDriver
- 4) Basics of Seetest Automation tool

Tool set required for Café Mobile Test Automation:

- 1) Eclipse (eg. Luna Release 4.4.0)
- 2) JDK 1.8, JRE 8.0
- 3) Seetest Automation tool (V8.5 and above)

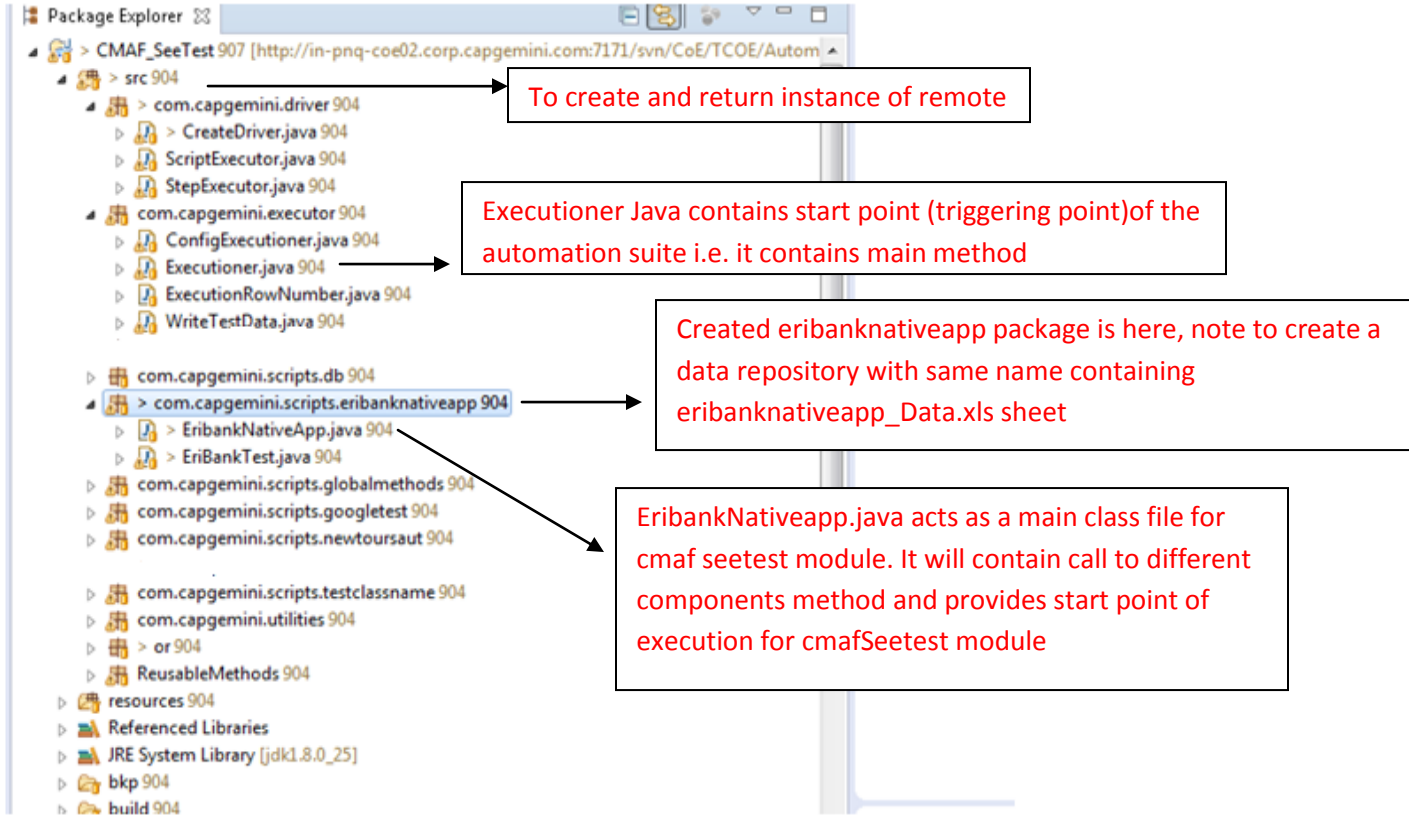
Setting Up Selenium with Café Mobile framework:

- 1) Extract The CAFÉ Mobile with SeeTest Zip file on your local machine or download it from the Testing COE SVN
- 2) Create a local workspace in Eclipse and import Selenium with SeeTest into it
- 4) Download and Install See test Automation tool (V8.5 and above)
- 3) Open the Selenium with SeeTest Project in eclipse and change view to project explorer view. It should look likes below :
Below highlighted are the core component of Selenium with SeeTest framework and it should be present before starting scripting.



Structure of the Project:

- **Src:** Folder contains all the java files where user should write all the core logic and perform scripting. Src folder contains different packages corresponding to each module and OR package.
 - A package corresponding to a module should contain all the java files associated with that particular module which take care of all the functionality associated with that module.
 - OR(Object Repository) Package should have Java file per web page and it should contain only element identification string(XATHs, IDs, CSSSelectors , Linktexts etc)
- **Referenced Libraries:** This Folder contains all the reference libraries for CAFÉ Mobile with SeeTest project which includes selenium JAR, POI XML JAR etc.
- **Data :** Folder contains all the data sheets(.xlsx files). It acts as a local data repository where all the referenced data is stored.
- **Configuration file:** Executioner.xlsx and Masterconfig.xlsx are the main configuration files for CMAF which configures the execution parameters.
- **Drivers:** Folder contains all the necessary driver files required to run the scripts on different browser.
- **Master Result:** Folder contains one MasterResult.xls sheet which will hold module wise test case count and module wise test case names. After completion of automation suite, folder contains one auto generated .xls sheet which will give detailed result of execution.
- **Results:** This folder contains html reports for the specific automation run. After completion of automation suite, module wise folder structure will be created through automation scripts to hold html reports. HTML reports will have step wise description of action performed along with link to the screenshot for that specific action.
- **Runnable:** Required in case of running automation from any CI tool or QC. It is auto generated after a set up for Ant is done
- Creating and Running the First Script:
 - Task – To automate Eribank Sample App
 - 1) Create a Package with respective module name in SRC package.
Suppose module name taken for the current task is Eribank App. So structure will be
src -> com.capgemini.scripts.eribanknativeapp
 - 2) Create a Java File with name as 'Eribanknativeapp' inside
com.capgemini.scripts.eribanknativeapp Package. It will be starting point to start execution for CAFÉ Mobile with Seetest module. Mandatory condition is package name and class name should be same(equalsignorcase)
 - 3) As part of Framework design, two packages com.capgemini.scripts.driver and com.capgemini.scripts.executor are created which contains core implementation of framework. Also com.capgemini.scripts.globalmethods package contains GlobalMethods.java which has methods to launch the applications, start the reporter time, and stop the reporter Time. It will look like below:



Package Explorer

> CMAF_SeeTest 907 [http://in-pnq-coe02.corp.capgemini.com:7171/svn/CoE/TCOE/Autom]

> src 904

> com.capgemini.driver 904

> CreateDriver.java 904

> ScriptExecutor.java 904

> StepExecutor.java 904

> com.capgemini.executor 904

> ConfigExecutioner.java 904

> Executioner.java 904

> ExecutionRowNumber.java 904

> WriteTestData.java 904

> com.capgemini.scripts.db 904

> com.capgemini.scripts.eribanknativeapp 904

> EribankNativeApp.java 904

> EriBankTest.java 904

> com.capgemini.scripts.globalmethods 904

> com.capgemini.scripts.googletest 904

> com.capgemini.scripts.newtoursaut 904

> com.capgemini.scripts.testclassname 904

> com.capgemini.utilities 904

> or 904

> ReusableMethods 904

> resources 904

> Referenced Libraries

> JRE System Library [jdk1.8.0_25]

> bkp 904

> build 904

To create and return instance of remote

Executioner Java contains start point (triggering point) of the automation suite i.e. it contains main method

Created eribanknativeapp package is here, note to create a data repository with same name containing eribanknativeapp_Data.xls sheet

EribankNativeapp.java acts as a main class file for cmaf seetest module. It will contain call to different components method and provides start point of execution for cmafSeetest module

- 4) In Eribanknativeapp.java, initialize the driver, start the reporter and Launch the applications, for which methods have already been defined at framework level.

So have to reuse our code for it:

DeviceName	Execute	URL	Mode	Platform	PlatformVersion	AppActivity
samsung SM-G900H	Yes	Eri Bank App	Seetest Cloud Device	Android	NA	com.experitest.ExperiBank.LoginActivity

Note : The Device Name is the name of the device as shown in the Seetest cloud device list

```

CreateDriver driver = new CreateDriver();
MobileWebDriver webDriver = driver.getWebDriver();

GlobalMethods globalMethods = new GlobalMethods(webDriver);

public String getExecutionStartTime(){
    return globalMethods.StrExecutionStartTime;
}

public String getStartTime(){
    return String.valueOf(globalMethods.executionStartTime);
}

```

Now GlobalMethods.LaunchApplication will launch the application as per execution parameters set in Executioner.xls file in Configuration Folder.

- ❖ In Executioner.xls, setup the option against each parameter whether to use it or not i.e. whether requirement is to run the test on DesktopBrowser or MobileBrowser or TabletBrowser or to Test Native App.
- ❖ A separate sheet for each platform (i.e. Desktop, Mobile, Tablet, native) is available which handles additional details about execution parameters such as on which all browsers test needs to run on, execution URL and associated Details.

E.g. Suppose in current scenario, requirement is to run test on native mobile application, so Executioner.xls setup will be like below:

Platform	Execute
NativeAppTest	Yes
TabletWebBrowser	No
MobileWebBrowser	No

- ❖ In MasterConfig.xls -> 'Classes' sheet, Make an entry for newly created module.

Note: Module Name should be exactly same as that of Package Name (i.e. ToursTravels) and mark it as 'Yes' in Execute Column so

that ToursTravels module will be triggered successfully.

Modules	Execute	Domain-Project	Execution Region
EriBankNativeApp	Yes		
TestClassName	No		

- ❖ Now Create a Data repository for EriBankNativeApp Module in Data folder.
So create a folder with name as EriBankNativeApp containing EriBankNativeApp_Data.xls data sheet.
So folder Structure will be like: Data -> EriBankNativeApp -> EriBankNativeApp_Data.xls
- ❖ In EriBankNativeApp.xls file, mandatory condition is to create first sheet with 'scenario' name. Now this scenario refers to different components (flows) which can be created as per the user need.

Component name mentioned in Scenario sheet should have corresponding method declared and defined in main class so that once user marks the component to Execute as 'YES' then scripts starts execution from method with same name in main Module Class file.

For current test, create a component as launch application, followed by, create a method as launchApplication in ToursTravels.java as below ->

```
public void launchApplication(){
    globalMethods.launchApplication();}
```

and scenario sheet look like as :

Scenario	Row	Execute
launchApplication	1	Yes

Now everything is setup and ready to start with first basic script to launch application. To trigger the application just run Src->com.capgemini.executor->Executioner.Java (as Executioner.Java contains main method)

After running the above test, you will find that a native application instance is opened and it will take up the TestCase which we have provided in Executioner sheet.

- ❖ Now Let's start with first test case to login into the application. Suppose we have test cases to automate as :
 - Test-case1:eriBankLogin
 - Test-case2: eriBankTest

For above test cases, let's create a component with name as LoginAndValidate. It needs basic two action

- i) To add entry in scenario sheet in EriBankNativeApp_Data.xls
- ii) Create a method in main module class i.e. create a method LoginAndValidate in EriBankNativeApp.Java file as below ->

Scenario	Row	Execute
launchNativeApplication	1	Yes
eriBankLogin	1	Yes

```
public void eriBanklogin(){}
```

Note: Make sure that access modifier should be public and method return type should be void only. Now all the core logic to perform login and validate a web page should be mention in eriBanklogin () method.

In Order to have a hierarchy of test cases and to achieve modularization it is recommended, to create a separate class which will handle all the processing of loginAndValidate method and perform operations on object of that class from eriBanklogin ().

```
public void eriBankLogin(){
    EriBankTest ebTest = new EriBankTest(webDriver);
```



```
        ebTest.eriBankLoginMethod();  
    }
```

and in testMethod Function, write a code to login into the application and to validate next Page.

Now to dynamically refer the data, make use of excel sheets. Create a separate sheet in EriBankNativeApp_Data.xls and store reference data in it.

Framework has provided lot of useful custom Packages which will help to complete given task with a great ease. Few of them are as below:

- **Utilities:** This package help us to perform various activities like to perform Excel operations, Generate report and perform operation on web elements. It contains -
 - Utility.Java : All Excel related I/O operations are present in this Class file.
 - CustomFunctions.Java : All stepwise operation like to select value from list, radio button status, text box operation, clicking on element all these operations are written in functions inside this file.
 - Reporter.Java : To generate HTML based report, this class provides required functions.
 - TestcaseReporter.Java : To generate Excel Based report to give overview of execution result.
- **Reusable Methods:** This Package help us to perform different validations and reusable actions.
 - CommonFunctions.Java : Programmer can write reusable methods like login, logout in this class.
 - VerificationFunctions.Java : Programmer can find lot of useful functions like validate text of particular element, presence of web elements on a web page in this class.

Now to progress with test, create one Page class file in Object Repository Package(OR) which will hold only element identification strings from a web page.(elements should be static and final as it should not be changed during scripts)

In current Example we will identify elements with Xpath as below :

```
package or;  
public class EriBankNativeApp {  
  
    public static final BUTTON_MAKEPAYMENT = "//*[@text='Make Payment'] |  
//*[@id='makePaymentButton']"  
}
```

Our source code will look like:

```

public void eriBankLoginMethod() {
    Reporter reporter = new Reporter();
    try{
        String strDataFileName =
utils.getDataFile("eriBankNativeTestData");
        reporter.start(reporter.calendar);

        int rowNumber = executionRowNumber.getExecutionRowNumber();
        String strPassword = scriptExecutor.readDataFile(strDataFileName,
rowNumber, "password");
        stepExecutor.enterTextValue("findElementByXPath",
EriBankNativeTest.TEXTBOX_USERNAME, strDataFileName, "username", webDriver);

        stepExecutor.enterTextValue("findElementByXPath",
EriBankNativeTest.TEXTBOX_PASSWORD, strDataFileName, "password", webDriver);

        stepExecutor.clickButton("findElementByXPath",
EriBankNativeTest.BUTTON_LOGIN, webDriver);
        stepExecutor.globalWait(2);
        objTestCaseReporter.reporter("TC_0");
        try{

            verificationFunctions.verifyElementPresent(webDriver, "xpath",
EriBankNativeTest.BUTTON_MAKEPAYMENT);
        }
        catch(Exception e)
        {
            objTestCaseReporter.reporter("TC_0");
            objTestCaseReporter.updateStatus("Fail");
        }

    }catch(Exception e1){
    }
}

```

Our Final Data Sheet looks like:

Scenario sheet:

Scenario	Row	Execute
launchNativeApplication	1	Yes
eriBankLogin	1	Yes

And Login sheet:

UserName	Password	Header
company	company	


Now you are ready to run test automation script using CAFÉ Mobile with SeeTest: Just trigger Executioner.java and observe the flow.

■ **How to Generate Reports :**

After the above run, HTML reports will get auto generated at below path :

Results ->EriBankNativeApp(Folder will get automatically generated)-> EriBankNativeApp _Report _yyyy-mm-dd-HH:MM.SS.MiliS format. (Generic values will be replaced by timestamp when execution run gets completed)


Complete HTML Report will looks like below:



Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

EriBankNativeApp Report					
Execution Date	Execution Start Time	Execution End Time	Total Execution Time	Operating System	Browser
19-11-2015	4:26:21 PM	4:26:43 PM	22 Second(s)	Windows 7	samsung SM-G900H
Scenario Name	Total Steps	Pass Steps	Fail Steps	Link to detail result	
LAUNCHNATIVEAPPLICATION	1	1	0	Detail Result	
ERIBANKLOGIN	13	13	0	Detail Result	

If we click on Detail result then detailed view will be provided as:



Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

EriBankNativeApp_eriBankTest Report						
Execution Date	Execution Start Time	Execution End Time	Total Execution Time	Operating System	Browser	
19-11-2015	4:26:28 PM	4:26:43 PM	15 Second(s)	Windows 7	samsung SM-G900H	
Test Scenario Name	Test Step Description	Test Data	Status	Result Description	ScreenShot	
ERIBANKLOGIN	Enter Value in text field	company	Pass	Value entered successfully	No Screenshot available	
ERIBANKLOGIN	Enter Value in text field	company	Pass	Value entered successfully	No Screenshot available	
ERIBANKLOGIN	Click on button	//*[@text='Login'] //*[@id='loginButton']	Pass	Clicked button successfully	No Screenshot available	
ERIBANKLOGIN	Verify element is present on the page	//*[@TEXT='MAKE PAYMENT'] //*[@ID='MAKEPAYMENTBUTTON']	Pass	Element is present on the page	No Screenshot available	
ERIBANKTEST	Click on button	//*[@text='Make Payment'] //*[@id='makePaymentButton']	Pass	Clicked button successfully	No Screenshot available	
ERIBANKTEST	Enter Value in text field	9999999999	Pass	Value entered successfully	No Screenshot available	
ERIBANKTEST	Enter Value in text field	Test	Pass	Value entered successfully	No Screenshot available	

Now Automation Programmer can analyze the progress of test scripts from above HTML results but it can't help us count of test cases passed and how many are failed/ in a No Run status.

This functionality can be achieved with Excel Reporter.

In current scenario, you have automated 3 test scripts corresponding to CAFÉ Mobile with SeeTest: Module.

So you have to add respective module name in 'ExecutionSummary' sheet with total number of test cases against it.

So Execution Summary Sheet looks like :

Scenario	Row	Execute		
launchNativeApp lication	1	Yes		
eriBankLogin	1	Yes	Pass	C:\Users\workspace\CMAF_SeeTest\results\EribankNativeApp\EribankNativeApp_launchNativeApplication_Report_2015-12-10-09.38.07.html
eriBankTest	1	Yes	Pass	C:\Users\workspace\CMAF_SeeTest\results\EribankNativeApp\EribankNativeApp_eriBankTest_Report_2015-12-10-09.38.30.html
quitApplication	1	Yes	Pass	
				-

Now, Trigger the Executioner.Java and validate the final outcome.

- ✓ HTML Reports can be found at → C:\Workspace\CMAF_SeeTest\results
- ✓ Excel Reports can be found at → C:\workspace\CMAF_SeeTest\EribankNativeApp_Data

In this way, you have automated test scripts and validated the results successfully through HTML and Excel based reports.

Mobile Test Automation Configuration

CAFÉ Mobile with SeeTest: Supports Cross Browser & Cross Platform Test Automation.

For Mobile Test Automation, it supports test automation for:

- Mobile Web Browsers
- Native Apps

It supports Automation over:

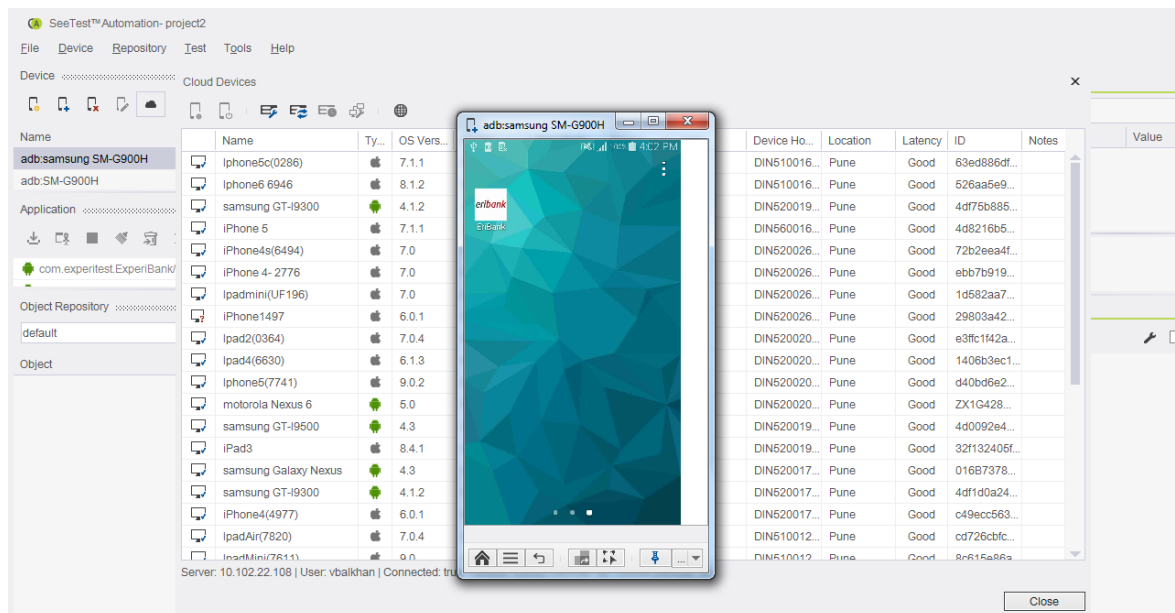
- Android Actual Devices
- Emulators
- iOS devices simulators
- iOS Actual Devices

Mobile NativeApp Test Execution on SeeTest Cloud Device:

- 1) Make sure the required application has been installed on the device
- 2) Also make sure that the Module has been added in the Native tab of the Materconfig.xls file and selected as yes
- 3) On the Executioner.xls make , platform tab only Native App should be selected and no other option should be selected
- 4) In the Native App tab add the device details with the correct application package details

C18													
	A	B	C	D	E	F	G	J	K	L	M	N	O
1	DeviceName	Execute	URL	Mode	Platform	PlatformVersion	AppActivity						
2	samsung SM-G900H	Yes	Eri Bank App	Seetest Cloud	Android	NA	com.experitest.ExperiBank.LoginActivity						
3	SM-T311	No	Eri Bank App	Actual Device	Android	NA	com.experitest.ExperiBank						

- 5) Open the Seetest Cloud Devices window in Seetest automation Tool.
- 6) Select the devices from the list of devices.
- 7) Open the device on which application has been installed.



- 8) Run Exectioner.java file.
- 9) Test Scripts execution will start on the selected open device.