Debugging Mobile Applications

The ColdFusion Mobile Platform aims at providing a server and development infrastructure that facilitates rapid and robust mobile application development, debugging, packaging, and deployment. The ColdFusion 11 release introduced rapid application development through ColdFusion Builder. ColdFusion 11 introduces full-fledged on-device debugging to quickly debug your ColdFusion-based mobile applications on devices.
The ColdFusion Builder has introduced a new debug server (agent) that acts as a broker between the IDE and the device. There will be a two-way communication between the debug agent and the IDE.

The IDE provides the debugging information like breakpoints, step over, step in, step out, and resume to the debug agent. In return, the debug agent provides information like where the code has paused to the IDE so that the IDE can show the appropriate source file with the highlighted line. When the device starts executing the code, communicates with the debug agent to get information on when to stop.

There are different scenarios through which you can debug your applications. ColdFusion Builder supports both local and remote debugging.

1. You can debug your application running on a browser.
2. You can debug a PhoneGap application running on the device.
3. You can debug a standalone HTML5-based mobile application running on the device.
4. Many developers can connect to a remote ColdFusion Server and debug their projects.

Debugging a packaged application

The new ColdFusion Builder enables you to run ColdFusion-based mobile applications interactively by inspecting the source code during the application execution. If you have already used Eclipse for debugging your applications, you will find debugging in ColdFusion builder easier.

While debugging your application on the browser or on a device, the following debug actions are supported:

1. You can set breakpoints (you can set handles in your source code specifying where the execution of your application must stop)
2. You can step in, step out, step over, and resume
3. You can get variables while debugging
4. You can set expressions while debugging

When your application completes loading/execution, you can investigate variables and change their content through a simple interface.

Verify the debug agent configuration

Before you start debugging your mobile application, verify the IP address and port of the Debug Agent. The IP address of the Debug Agent is most likely the IP address of your machine. These IP address and the port value will be automatically configured. To verify or change these values, click Windows > Preferences > ColdFusion > Client Debug Settings.

Setting breakpoints

To set breakpoints in your source code right-click in the small left margin in your source code editor and select Toggle Breakpoint. Alternatively, you can also double-click on this position to enable or disable breakpoints.

Starting the debugger

You need to start the debugger to start debugging ColdFusion mobile applications. Perform the following tasks:

- Before you begin, you need to generate the PhoneGap Debug Build if you are planning to debug your applications on the device. Right click your project and select Generate PhoneGap Debug Build. If you have not configured your PhoneGap settings, you will be prompted to enter the PhoneGap account settings.

Once you generate the PhoneGap build, you need to transfer the binary file to your device and install the application. For instance, on Android, you transfer the APK file to the device and install the APK file.

Start the debugger by clicking the Debug Icon on the menu and selecting Debug As > ColdFusion Client Applications

Depending on where you have set the breakpoint, your application may look blank when it starts on your device:

- At this moment, you can see the control at the breakpoint position in ColdFusion Builder. Continue stepping over and debug your application.

This action will start the device debugger. If you have not defined any breakpoints, this action will run your application normally. To debug the application, you need to define breakpoints.

Note: If the code being executed on the browser or the shell application encounters a runtime error, you will see an error message and the debugger will fail to start.

Viewing variables and evaluating expressions

While the debugging session is on, you can view the variables and evaluate expressions easily. Ensure that you have activated the ColdFusion Debugging perspective:

- Click Window > Open Perspective > Other.. and select ColdFusion Debugging:
- Click OK to continue
- You can start viewing the variables through the following window:
- Similarly, you can evaluate expressions:

Stepping in and out

ColdFusion Builder allows you to control the execution of the application you are debugging through simple code stepping actions.

You can step in, step out, and step over to view values of variables and can evaluate expressions.

Debugging a shell application

Setting breakpoints

To set breakpoints in your source code right-click in the small left margin in your source code editor and select Toggle Breakpoint. Alternatively, you can also double-click on this position to enable or disable breakpoints.

Note that the breakpoints will not be hit for JavaScript code in your application.

Starting the debugger

You need to start the debugger to start debugging ColdFusion mobile applications. Perform the following tasks:

- Start the debugger by clicking the Debug Icon on the menu and selecting Debug As > ColdFusion Client Applications
- Invoke the application URL from the device using the ColdFusion Shell application:

- Depending on where you have set the breakpoint, your application may look blank when it starts on your device:
- At this moment, you can see the control at the breakpoint position in ColdFusion Builder. Continue stepping over and debug your application.

This action will start the device debugger. If you have not defined any breakpoints, this action will run your application normally. To debug the application, you need to define breakpoints

---

Viewing variables and evaluating expressions

While the debugging session is on, you can view the variables and evaluate expressions easily. Ensure that you have activated the ColdFusion Debugging perspective:

- Click Window > Open Perspective > Other.. and select ColdFusion Debugging:
- Click OK to continue
- You can start viewing the variables through the following window:
- Similarly, you can evaluate expressions:

Stepping in and out

ColdFusion Builder allows you to control the execution of the application you are debugging through simple code stepping actions.

You can step in, step out, and step over to view values of variables and can evaluate expressions.

Test Automation of Mobile App

Mobile App Testing help you automate testing of your Android and iOS Apps. These Mobile Application testing software can reduce the time needed for the testing process and the chances of human errors during test execution.Following is a handpicked list of Top Mobile Automation Tools , with their popular features and download links. The list contains both open source(free) and commercial(paid) software.

1) Kobiton

Kobiton gives users full control of real mobile devices during manual testing with support for multi-touch gestures, orientation and GPS simulations, camera and speaker control and device connection management. With automatically generated activity logs, Kobiton captures all the actions performed during a testing session so issues can be identified and resolved more quickly. Users can purchase prepaid testing minutes that never expire for just $10.

Benefits:

- The latest real, cloud-based devices and configurations
- Centralized testing history and data logs for increased collaboration
- Internal Device Lab Management to more effectively utilizes internal devices
- Support for Appium 1.6.4
- Simplified user experience to streamline test sessions
- Easy to try with Free Trial – no credit card required

2) TestProject

TestProject is the world's first free cloud-based, a community-powered test automation platform that enables users to test Web, Android and iOS applications on all operating systems, effortlessly. Easily collaborate with your team using Selenium and Appium to ensure quality with speed. Use advanced built-in recording capabilities, create and use addons (automation actions shared by the entire community), or develop coded tests using TestProject's powerful SDK, all completely for FREE!

- No complex setups or configurations
- No coding skills required to get started
- Share and reuse addons with your team and the entire community
- Detailed report dashboards
- Seamless integrations with your CI/CD workflow

3) FrogLogic

Solve your Mobile test automation challenges with Squish for iOS and Squish for Android. Squish features dedicated support for automated testing of native Mobile Apps, mobile Web Apps as well as a mixture of both.

Due to Squish's unique and stable object identification methods, Squish tests can run on mobile device emulators and different real devices without any changes. Unlike many other test tools, Squish does not require you to jailbreak or root the device. Instead, you can get started with automated GUI testing of your mobile Apps right away!

Benefits:

- Advanced gesture support
- Ready for Testing in the Cloud
- Support for embedded web content
- CI and source control integration
- End-to-End and IoT testing

4) TestingBot

TestingBot provides real mobile device testing in the cloud. Run automated and manual tests on physical Android and iOS devices in TestingBot's device farm.

- Support for latest Appium and Selenium versions
- Integrates with your CI/CD pipelines
- Live interaction with physical iOS and Android devices
- Test on older devices, new devices and upcoming beta versions
- Access to screenshots, video and various other metrics of the device during testing
- Starts at $49 per month for unlimited mobile testing.

5) Apptim

Apptim empowers mobile developers and testers to easily test their apps and analyze their performance in each build to prevent critical issues from going live. Measure app render times, power consumption, resource usage, capture crashes, errors, and more on Android and iOS devices.

- Android and iOS compatible

- Easily troubleshoot app crashes and exceptions
- Compare app performance of two different builds to find out what changed and identify potential new performance issues
- Integrates with JIRA out of the box and your workflow
- Free to use
- Get your first results in just 5 minutes

6) HeadSpin

HeadSpin provides real-world, actionable user experience insights for businesses to improve mobile performance. HeadSpin prepares you for the increasing challenges in dealing with customer experiences across the complex mobile ecosystem by providing detailed visibility into performance and user experience issues across every layer of the mobile stack - from client to server.

Features:

- Thousands of real devices in 150+ locations around the world (No device jailbreaking, rooting, no virtual machines)
- No SDK for you to expand your code base. All via API access.
- AI engine automatically organizes performance issues from largest to smallest time impact
- Pre- and post-launch visibility
- Cloud and on-premise setup available
- Dedicated, shared or pay-as-you-go access available
- Load testing available
- 100% uptime on devices

8) Selendroid

Selendroid is a test automation framework that drives off the UI of Android native and hybrid applications (apps) and the mobile web. Using the Selenium 2 client API tests are written.

Benefits of Selendroid

- It is fully compatible with JSON wire protocol
- No alteration of app under test is needed to automate it
- Same concept for automating native or hybrid apps
- By different locator types, UI elements can be found
- It can interact with multiple Android devices at the same time
- Selendroid supports hot plugging of hardware devices
- By different locator types, UI elements can be found

Selendroid comes with a useful tool known as Selenium Inspector. It allows you to inspect the current state of your app's UI.

White Box Testing and Black Box Testing

Software testing is the process of evaluating software functionality and quality by detecting bugs and later removing them with the help of QA team or an efficient testing tool.

Testing validates a software by checking whether it is meeting business and technical requirements with guided design.

Software Testing is required for the following reasons:

- Cost Effective
- Security
- Product Quality
- Customer Satisfaction

There are several different testing techniques; Black Box and White Box testing are two such approaches commonly used by testers.

Black Box Testing

Testing is broadly based on software requirements and specifications. Black Box Testing is a technique in which tester is unaware about the internal structure or code of the software.

The focus is on inputs and outputs ignoring the internal knowledge of the code. Using black box testing, one can test operating systems like Windows, websites like Google and even our own customized applications, as the core knowledge about these operating systems are not required.

How Black Box Testing Works?

Black box testing can broadly be summarized into the following steps.

1. The first step is to thoroughly examine the requirements and specifications of the system.
2. The tester explores the system’s UI and functionality to understand how the processes on the system are expected to work.
3. On later stage, the tester checks efficiency of the software by determining expected outputs with their corresponding inputs.
4. Finally, the developer fixes the bug detected and the output undergoes retesting.

Black Box Testing Techniques

There are three techniques usually employed by organizations and testers in case of Black Box Testing.

Equivalent Class Testing: It is used to reduce the number of possible test cases to an ideal level to maintain a reasonable test coverage.

Boundary Value Testing: It determines whether certain range of values are accepted by the software or not. This helps in reducing number of test cases.

Decision Table Testing: A decision table puts conditions and their outcomes in a matrix. There is a unique combination in every segment.

Advantages

- Suitable for large code segments
- Increased Efficiency
- Prior knowledge of code is not required

Black box testing is all about enhancing the user experience even if they are from a non-technical background. On the other hand, for technical support and precise coding, White box testing is an excellent approach for organizations to employ. Let's understand the nitty gritty of what goes behind White Box Testing.

White Box Testing

White Box Testing is also known as open, transparent or glass box testing. In white box testing, the tester has prior knowledge of the code and accordingly prepares the test case.

The tester has the knowledge of the internals of a system and knows how the system is implemented. The tester uses this knowledge to develop test cases that will examine the control flow, information flow, data flow, exception and error handling as well as coding practices of the system.

How does White Box Testing work?

Here's how White Box Testing works ...

1. The first step for the tester is to understand the source code.
2. White Box testing then involves testing of internal functions of the application, so knowledge of source code is crucial.

3. The tester should be aware of the secure coding practices as security is the most important factor in testing.
4. Tester can then write code for testing the application or can prepare certain test cases with suitable inputs.

White Box Testing Techniques

<u>Code Coverage Analysis:</u> It eliminates gaps in test case suite by identifying the program which cannot be examined by test cases. In addition, you can create test cases for untested part of the program which improves the quality of the software.

<u>Statement Coverage:</u> This technique checks every statement of the code at least once during the test cycle.

<u>Branch Coverage:</u> This technique tests every possible path in the code like If-else loops and other conditional loops of the software.

Advantages

- It optimizes the code as it tests every statement of the code.
- Automated testing is supported.
- Tests and test scripts can be reused.
- Testing is supported at early development stages.

Software testing is the most important part for maintaining the quality of the software. Manual and automated testing both are required to test the software thoroughly.

Taking up the black box and white box testing using an automated testing tool such as TestingWhiz is highly recommended. Automated testing allows the tester to focus more on the high priority issues plaguing on the deployment of the software instead of the repetitive mundane tasks that testing needs. This helps save time, increase productivity and efficiency of the testers along with boosting employee morale. Moreover, testers from all backgrounds can use it seamlessly.

JUnit

JUnit is a unit testing framework for Java programming language. JUnit has been important in the development of test-driven development, and is one of a family of unit testing frameworks collectively known as xUnit, that originated with JUnit.

Unit testing can be done in two ways − manual testing and automated testing.

Manual Testing	**Automated Testing**
Executing a test cases manually without any tool support is known as manual testing.	Taking tool support and executing the test cases by using an automation tool is known as automation testing.
Time-consuming and tedious − Since test cases are executed by human resources, it is very slow and tedious.	Fast − Automation runs test cases significantly faster than human resources.
Huge investment in human resources − As test cases need to be executed	Less investment in human resources − Test cases are executed using automation tools,

manually, more testers are required in manual testing.	so less number of testers are required in automation testing.
Less reliable − Manual testing is less reliable, as it has to account for human errors.	More reliable − Automation tests are precise and reliable.
Non-programmable − No programming can be done to write sophisticated tests to fetch hidden information.	Programmable − Testers can program sophisticated tests to bring out hidden information.

What is JUnit ?

JUnit is a unit testing framework for Java programming language. It plays a crucial role test-driven development, and is a family of unit testing frameworks collectively known as xUnit.

JUnit promotes the idea of "first testing then coding", which emphasizes on setting up the test data for a piece of code that can be tested first and then implemented. This approach is like "test a little, code a little, test a little, code a little." It increases the productivity of the programmer and the stability of program code, which in turn reduces the stress on the programmer and the time spent on debugging.

Features of JUnit

- JUnit is an open source framework, which is used for writing and running tests.
- Provides annotations to identify test methods.
- Provides assertions for testing expected results.
- Provides test runners for running tests.
- JUnit tests allow you to write codes faster, which increases quality.
- JUnit is elegantly simple. It is less complex and takes less time.
- JUnit tests can be run automatically and they check their own results and provide immediate feedback. There's no need to manually comb through a report of test results.
- JUnit tests can be organized into test suites containing test cases and even other test suites.
- JUnit shows test progress in a bar that is green if the test is running smoothly, and it turns red when a test fails.

JUnit is a framework for Java, so the very first requirement is to have JDK installed in your machine.

Step 1: Verify Java Installation in Your Machine

Step 2: Set JAVA Environment

Step 3: Download JUnit Archive

Step 4: Set JUnit Environment

Step 5: Set CLASSPATH Variable

Step 6: Test JUnit Setup

Step 7: Verify the Result