Adobe ColdFusion Documentation

September 2014

Mobile Application Development

Building Mobile Applications

Overview

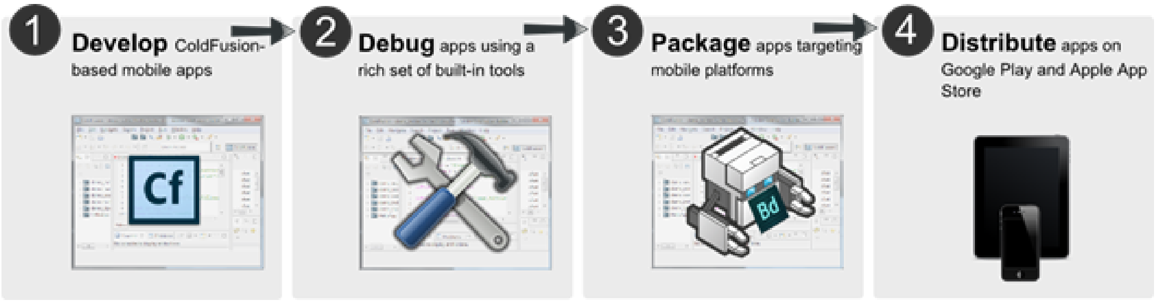
Mobile is soon turning in to one of the most commonly applicable targets for software development. When budgeting for application development, significant investment is reserved for the mobile platform since smartphones and, more recently, tablets are increasingly becoming the choice of social interaction for more and more people. If you have been already developing mobile applications, you will know the effort involved in understanding and targeting various mobile platforms.

Using **ColdFusion 11** and **ColdFusion Builder 3** (comprising the ColdFusion Mobile Platform), you can quickly build mobile applications targeting multiple platforms. You do not have to focus on the limitations and capabilities of various mobile platforms and their supporting web browsers. You can build your mobile applications completely using ColdFusion and can make it work equally well on desktop browsers, mobile browsers, and as installed HTML5-based native 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 platform comprises of the following key components:

- **The CFML Language**: Added capabilities to the CFML programming language that facilitates conversion of CFML code to mobile-friendly HTML/JavaScript, enabling easy adoption of newer web standards such as HTML5.
- **Integration with PhoneGap framework:** For quickly building cross-platform mobile applications using ColdFusion-translated JavaScript.
- **ColdFusion Builder:** You can use ColdFusion Builder along with ColdFusion Server for quickly building, debugging, inspecting, and packaging mobile applications.

The following picture depicts the mobile application development workflow using the ColdFusion Mobile Platform:

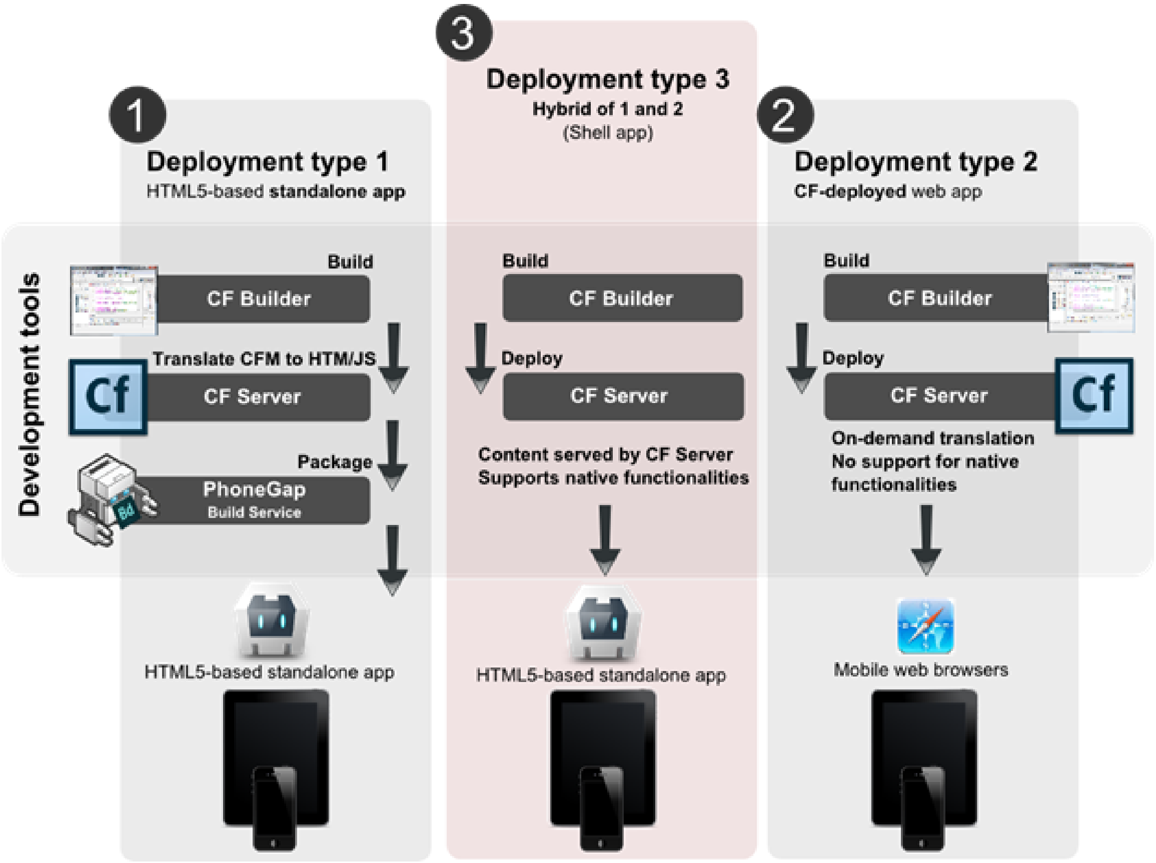

1 Develop ColdFusion-based mobile apps
2 Debug apps using a rich set of built-in tools
3 Package apps targeting mobile platforms
4 Distribute apps on Google Play and Apple App Store
Cf
Bd

With the ease of programming that CFML facilitates and with the power of JavaScript, the ColdFusion Mobile platform will be a robust platform and a complete solution for mobile applications development.
Leveraging the ColdFusion Mobile platform, you can rapidly develop mobile applications in CFML. Note that the CFML constructs relevant for client-side applications development will be converted to JavaScript automatically by ColdFusion. This means that not all CFML features, functions, tags, and other functionality are supported for conversion to JavaScript. See Debugging mobile applicationsClient-side CFML for a list mobile-enabled tags and functions.
Note: This version of the pre-release bundle supports debugging while building mobile applications using ColdFusion Builder.

Types of mobile applications

ColdFusion Mobile Platform allows you to create 3 types of mobile applications:

- **Type 1 - HTML5-based standalone app** that can be installed on the mobile device as a standalone application. ColdFusion Builder allows you to build platform-specific mobile applications using Adobe's Phone Gap build service.
- **Type 2 - ColdFusion-deployed web app** that can be rendered through the mobile device's web browser. There is no support for device's native functionalities.
- **Type 3 – Hybrid/shell app** that can run as a standalone mobile application. This type supports device's native functionalities.



In the case of Type 1, platform-specific PhoneGap JavaScript file and the supporting native libraries are bundled with the application. In the case of Type 2, mobile browser or any other web browser can be used to access ColdFusion applications. In Type 3 (hybrid app or shell app), only the native libraries are bundled. The PhoneGap JavaScript files and the application content gets served by the ColdFusion server.
You can decide on the type of application based on the application requirements.

Note: You can use the window.ispgbuild property to identify if an application is running as shell application or packaged application.

Configuring the development environment

Setting up the ColdFusion server

Install ColdFusion Server 11 by following the instructions provided in the installation guide. While installation, you do not need to configure the server to support mobile applications as the support is available by default.

Setting up ColdFusion Builder

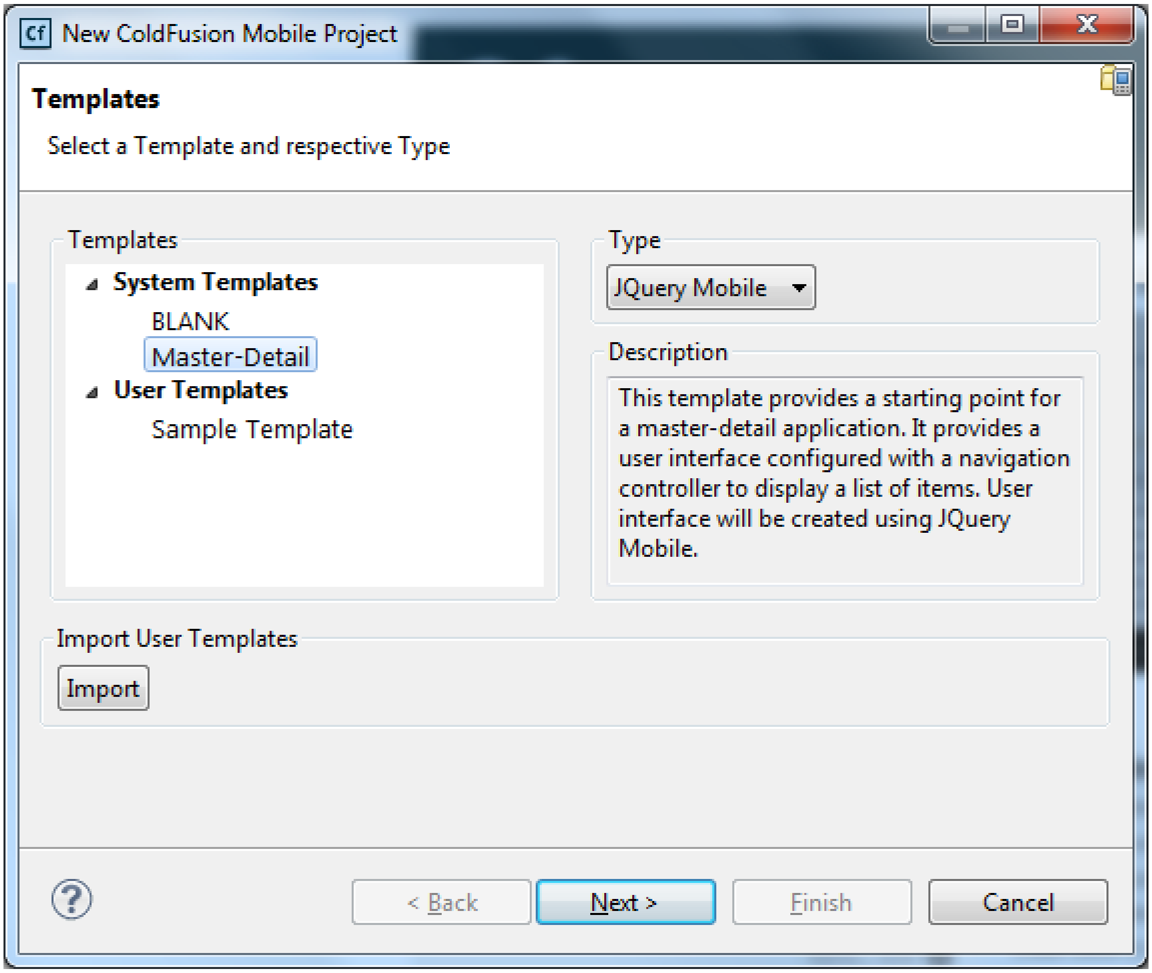
Install ColdFusion Builder by following the instructions provided in the Installing Adobe ColdFusion Builder guide. Before you start reading this document, read the Using Adobe ColdFusion Builder guide.

Creating a mobile project

Note that ColdFusion Builder has introduced a new '**ColdFusion Mobile Project**' type that will help you build and package mobile applications. However, you can convert your existing ColdFusion projects to a ColdFusion Mobile projects automatically. See Migrating existing projects.

To create a ColdFusion Mobile project, perform the following tasks:

1. Click **File** > **New** > **ColdFusion Mobile Project** or Right-click on the Navigator area and click **New** > **ColdFusion Mobile Project**. You will be asked to select a mobile template. For more information on creating your own mobile template, read Mobile Templates. To import existing templates, click Import. Note that the mobile template must conform to the standards outlined in Mobile Templates.



New ColdFusion Mobile Project

Templates

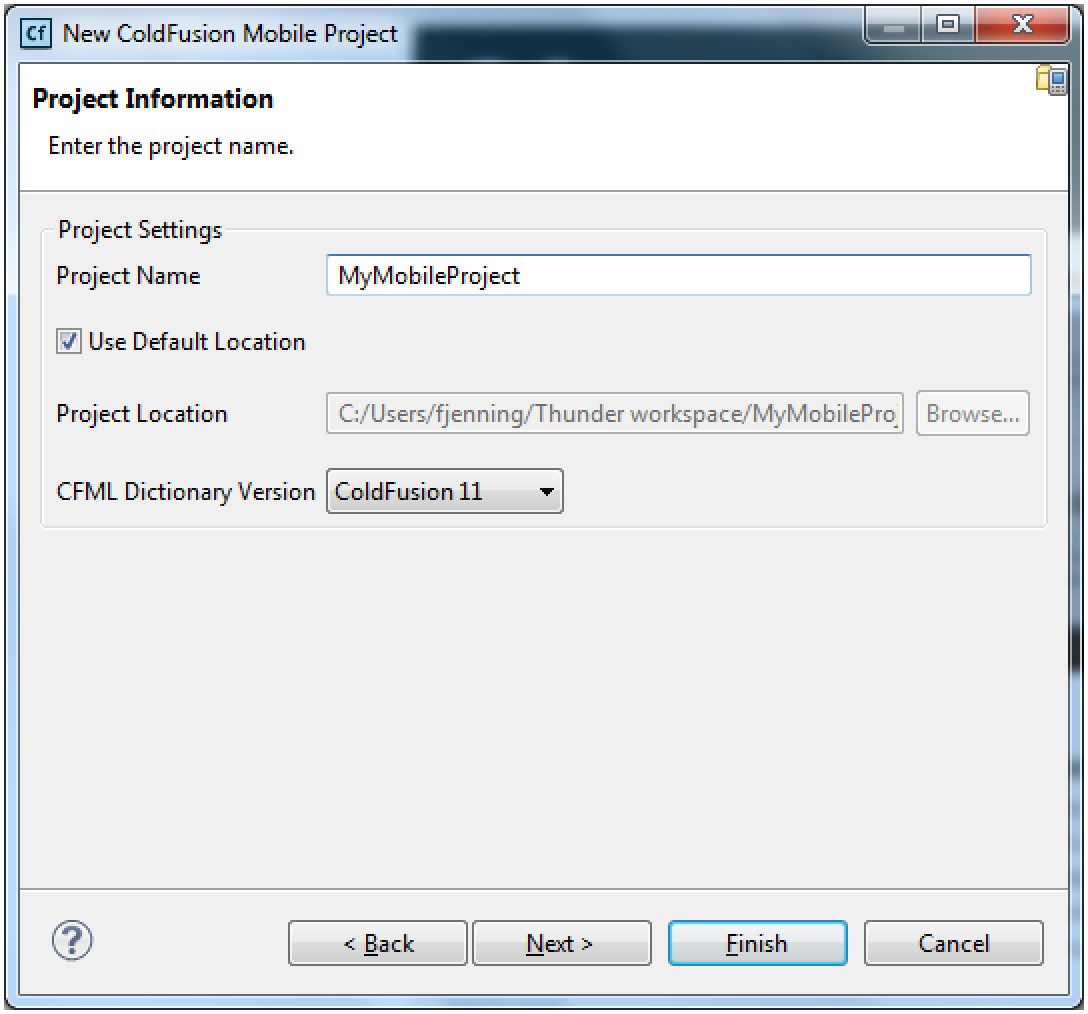
Select a Template and respective Type

Templates
- **System Templates**
  - BLANK
  - Master-Detail
- **User Templates**
  - Sample Template

Type: JQuery Mobile

Description: This template provides a starting point for a master-detail application. It provides a user interface configured with a navigation controller to display a list of items. User interface will be created using JQuery Mobile.

2. Click **Next**.
3. In the Project Information section, enter the following details:
   a. Enter the name of your project.
   b. Select the CFML Dictionary Version as **ColdFusion 11**.



New ColdFusion Mobile Project

Project Information

Enter the project name.

Project Settings

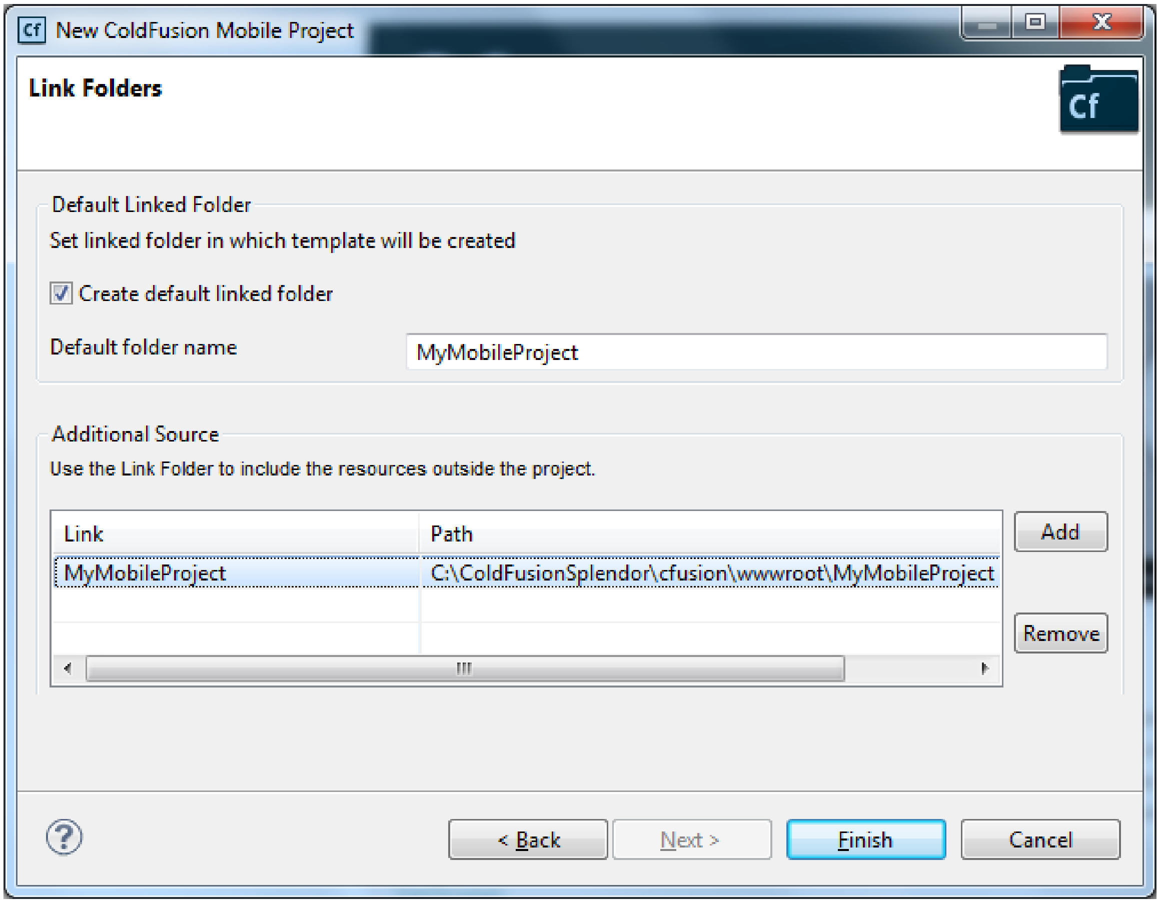
Project Name: MyMobileProject

☑ Use Default Location

Project Location: C:/Users/fjenning/Thunder workspace/MyMobilePro

CFML Dictionary Version: ColdFusion 11

4. Click **Next**.
5. In the **Server Details** screen, from the **Servers** dropdown box, and click **Add Server**.
6. In the **Server Settings** screen, enter the server details. For more information, read the Using ColdFusion Builder guide. Click **Next**.
7. In the Link Folders section, map the specific ColdFusion directories available under the Server's web root to your project:



New ColdFusion Mobile Project

Link Folders

Default Linked Folder

Set linked folder in which template will be created

☑ Create default linked folder

Default folder name: MyMobileProject

Additional Source

Use the Link Folder to include the resources outside the project.

Link	Path
MyMobileProject	C:\ColdFusionSplendor\cfusion\wwwroot\MyMobileProject

8. Click **Finish**.

You have successfully created a ColdFusion Mobile project. Now you can start building your mobile applications. Note that at this point, you do not need to configure the project with any PhoneGap settings.

Write some code

Now that you have a mobile project, write some code to capture an image from the device camera:

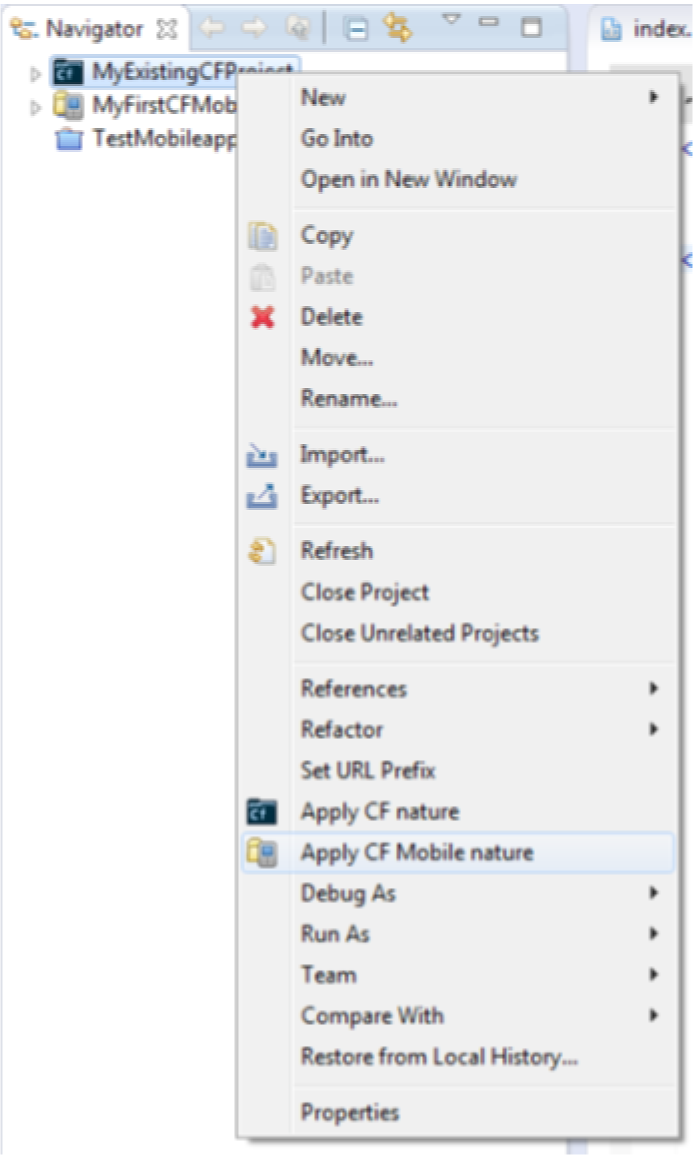
```
<cfclientsettings enableDeviceAPI=true>

<cfclient>
    <cfset
document.getElementById('result').innerHTML=document.getElementById('result').innerH
TML+"Hello from cfclient!">
</cfclient>

<div id="result"/>
```

To know more about writing client-side CFML, see the following sections:

- Client-side CFML (for mobile development)
- Getting Started Examples

Migrating existing projects

You can automatically convert your existing ColdFusion projects to ColdFusion Mobile projects by applying the ColdFusion Mobile nature as shown in the following screen shot:



Examples

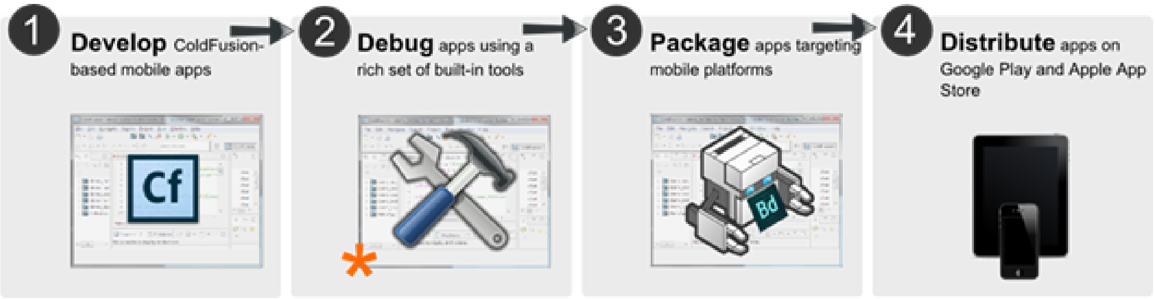
See Getting Started Examples.

Where to go from here

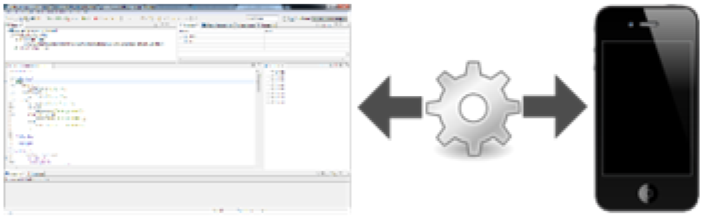
- Learn the ColdFusion Mobile Functions
- Cleint-side CFML - **Before building** your ColdFusion-based mobile application, learn the CFML tags and functions that will enable you to build simple to complex mobile applications.
- Packaging mobile applications – **After building** your ColdFusion-based mobile application, you can package the application into platform-specific installers.

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.


1 Develop ColdFusion-based mobile apps
2 Debug apps using a rich set of built-in tools
3 Package apps targeting mobile platforms
4 Distribute apps on Google Play and Apple App Store
Cf
Bd
*

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.

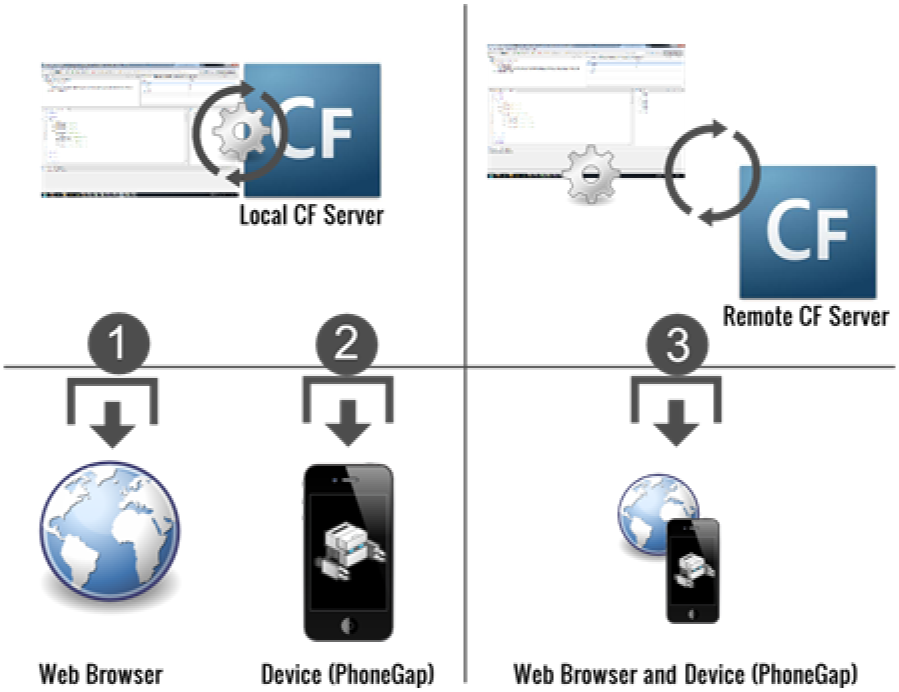


The ColdFusion Builder Debug Agent intercepts and processes debug commands from the IDE and device

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.



In scenario 3, developers cannot debug the same ColdFusion mobile project simultaneously.

Debugging a packaged application

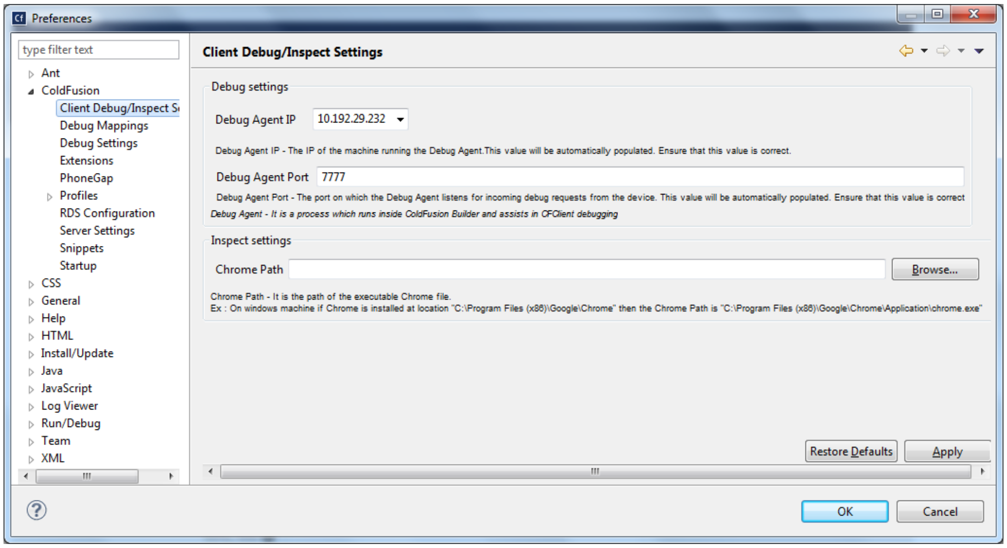
The new ColdFusion Builder enables you to run ColdFusion-based mobile applications interactively by inspecting the source code during the application execution. If you have already used Eclipse for debugging your applications, you will find debugging in ColdFusion builder easier.
While debugging your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



Client Debug/Inspect Settings

Debug settings

Debug Agent IP 10.192.29.232

Debug Agent IP - The IP of the machine running the Debug Agent.This value will be automatically populated. Ensure that this value is correct.

Debug Agent Port 7777

Debug Agent Port - The port on which the Debug Agent listens for incoming debug requests from the device. This value will be automatically populated. Ensure that this value is correct

Debug Agent - It is a process which runs inside ColdFusion Builder and assists in CFClient debugging

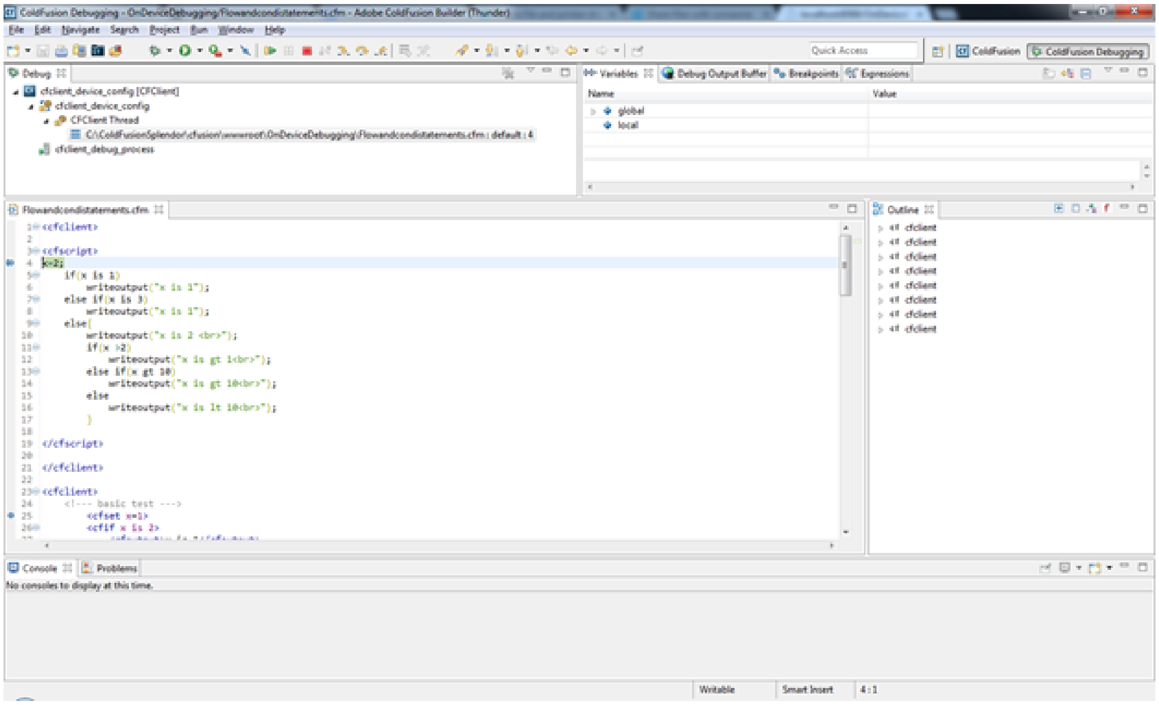
Inspect settings

Chrome Path

Chrome Path - It is the path of the executable Chrome file.
Ex : On windows machine if Chrome is installed at location "C:\Program Files (x86)\Google\Chrome" then the Chrome Path is "C:\Program Files (x86)\Google\Chrome\Application\chrome.exe"

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.



ColdFusion Debugging - OnDeviceDebugging/Flowandcondistatements.cfm - Adobe ColdFusion Builder (Thunder)
File Edit Navigate Search Project Run Window Help
Quick Access
ColdFusion
ColdFusion Debugging
Debug
Variables
Debug Output Buffer
Breakpoints
Expressions
Name
Value
Flowandcondistatements.cfm
Outline
Console
Problems
Writable
Smart Insert
4 : 1

cfclient_device_config [CFClient]
cfclient_device_config
CFClient Thread
C:\ColdFusionSplendor\cfusion\wwwroot\OnDeviceDebugging\Flowandcondistatements.cfm : default : 4
cfclient_debug_process

global
local

cfclient
cfclient
cfclient
cfclient
cfclient
cfclient
cfclient
cfclient

```
<cfclient>

<cfscript>
x=2;
    if(x is 1)
        writeoutput("x is 1");
    else if(x is 3)
        writeoutput("x is 1");
    else{
        writeoutput("x is 2 <br>");
        if(x >2)
            writeoutput("x is gt 1<br>");
        else if(x gt 10)
            writeoutput("x is gt 10<br>");
        else
            writeoutput("x is lt 10<br>");
        }

</cfscript>

</cfclient>

<cfclient>
    <!--- basic test --->
        <cfset x=1>
        <cfif x is 2>
```

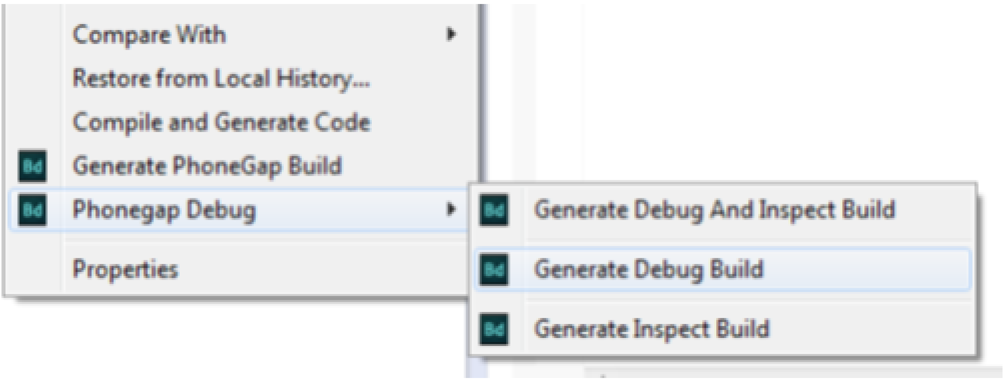
No consoles to display at this time.

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

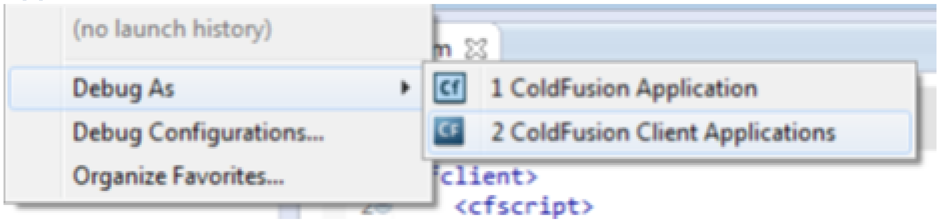
Starting the debugger

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

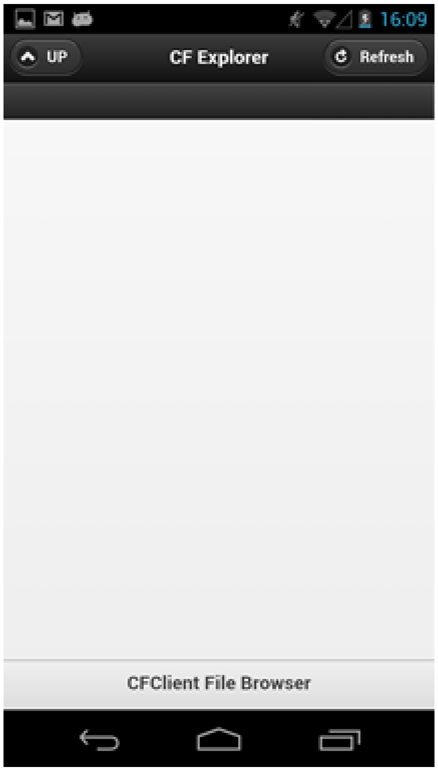
1. Before you begin, you need to generate the PhoneGap Debug Build if you are planning to debug your applications on the device. Right click your project and select Generate PhoneGap Debug Build. If you have not configured your PhoneGap settings, you will be prompted to enter the PhoneGap account settings.



2. 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.
3. Start the debugger by clicking the Debug Icon on the menu and selecting **Debug As > ColdFusion Client Applications**



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



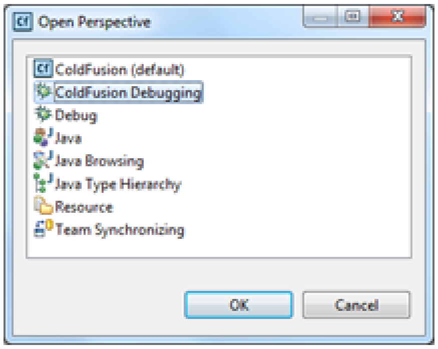
5. At this moment, you can see the control at the breakpoint position in ColdFusion Builder. Continue stepping over and debug your application.

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.

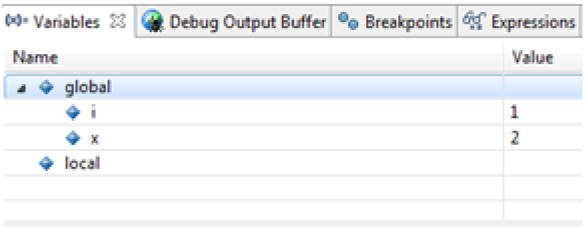
Viewing variables and evaluating expressions

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

1. Click Window > Open Perspective > Other.. and select ColdFusion Debugging:

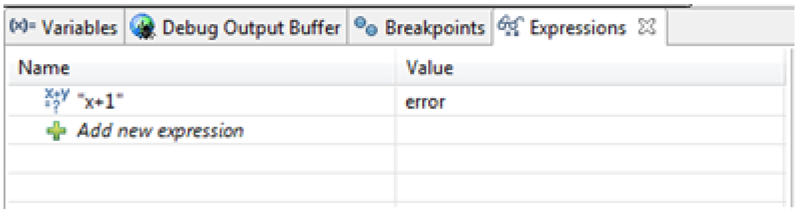


2. Click OK to continue
3. You can start viewing the variables through the following window:



Name	Value
global	
i	1
x	2
local	

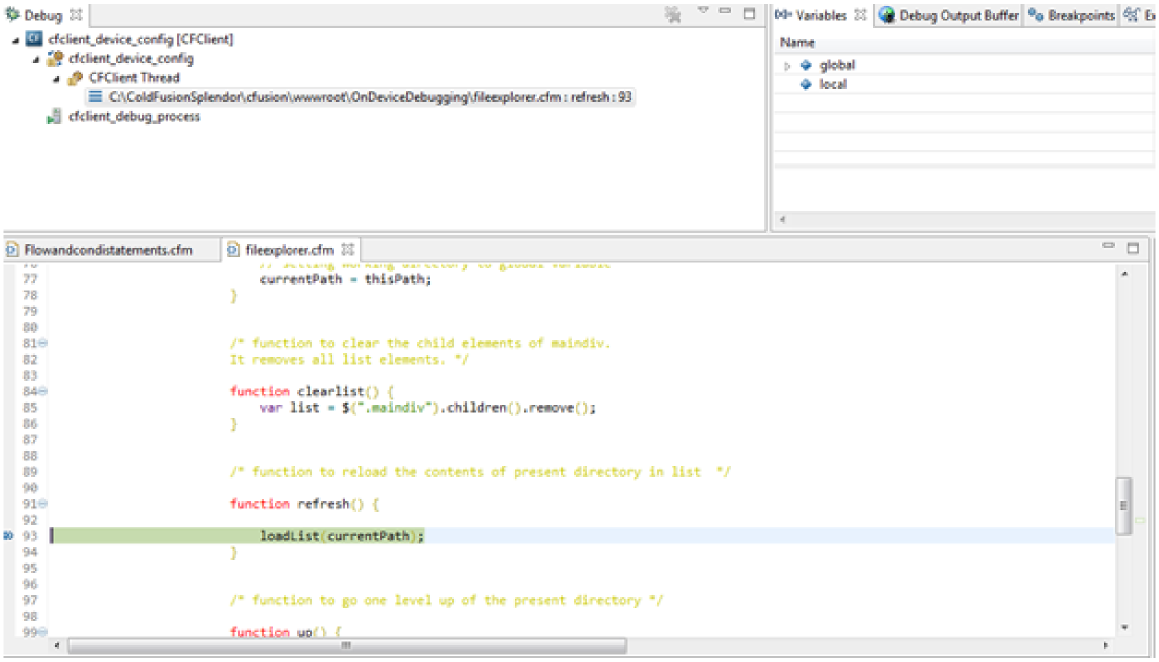
4. Similarly, you can evaluate expressions:



Name	Value
"x+1"	error
Add new expression	

Stepping in and out

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



```
cfclient_device_config [CFClient]
  cfclient_device_config
    CFClient Thread
      C:\ColdFusionSplendor\cfusion\wwwroot\OnDeviceDebugging\fileexplorer.cfm : refresh : 93
  cfclient_debug_process
```

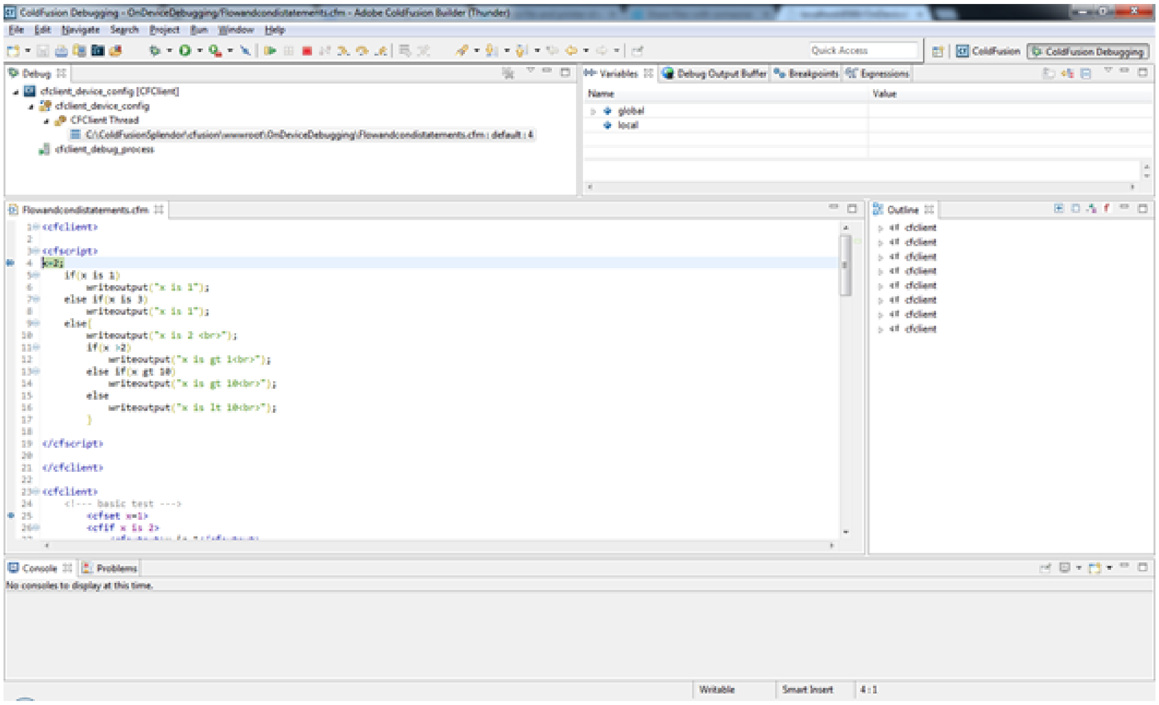
```
                currentPath = thisPath;
            }


            /* function to clear the child elements of maindiv.
            It removes all list elements. */

            function clearlist() {
                var list = $(".maindiv").children().remove();
            }


            /* function to reload the contents of present directory in list  */

            function refresh() {

                loadList(currentPath);
            }


            /* function to go one level up of the present directory */

            function up() {
```

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.



```
cfclient_device_config [CFClient]
  cfclient_device_config
    CFClient Thread
      C:\ColdFusionSplendor\cfusion\wwwroot\OnDeviceDebugging\Flowandcondistatements.cfm : default : 4
  cfclient_debug_process
```

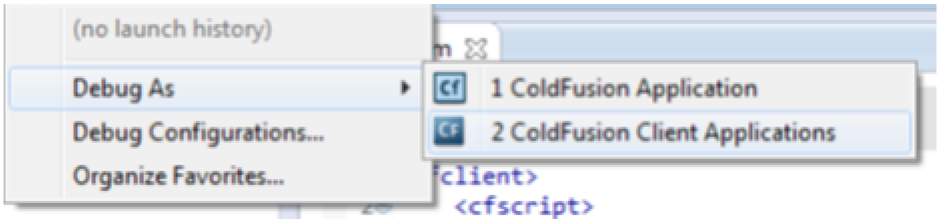
```
<cfclient>

<cfscript>
x=2;
    if(x is 1)
        writeoutput("x is 1");
    else if(x is 3)
        writeoutput("x is 1");
    else{
        writeoutput("x is 2 <br>");
        if(x >2)
            writeoutput("x is gt 1<br>");
        else if(x gt 10)
            writeoutput("x is gt 10<br>");
        else
            writeoutput("x is lt 10<br>");
        }

</cfscript>

</cfclient>

<cfclient>
    <!--- basic test --->
        <cfset x=1>
        <cfif x is 2>
```

No consoles to display at this time.

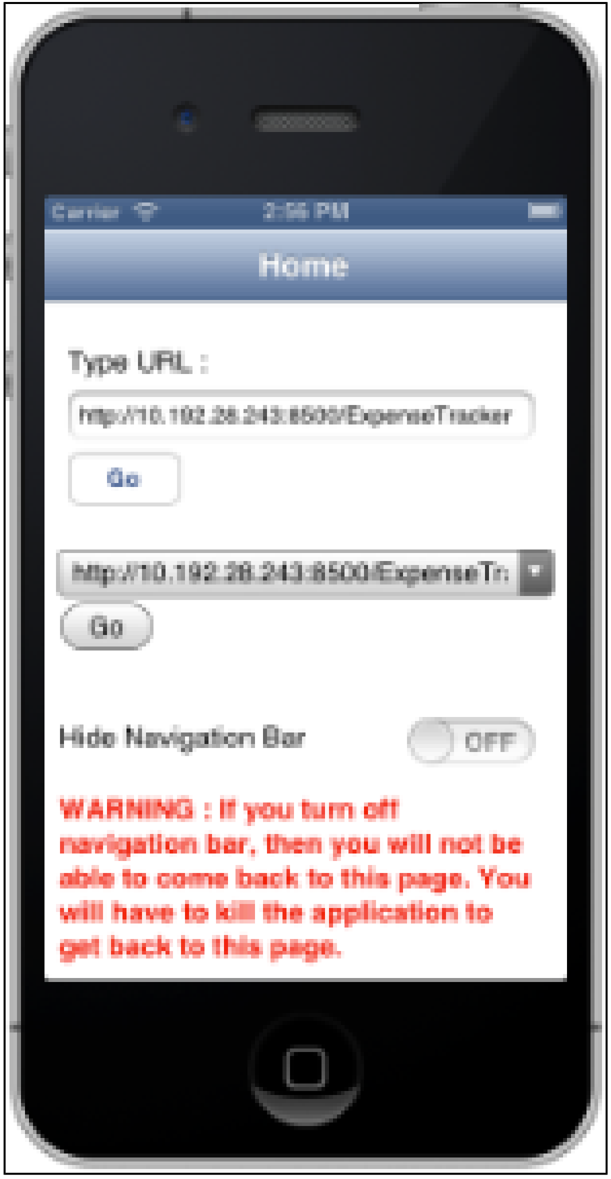
Starting the debugger

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

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



2. Invoke the application URL from the device using the ColdFusion Shell application:



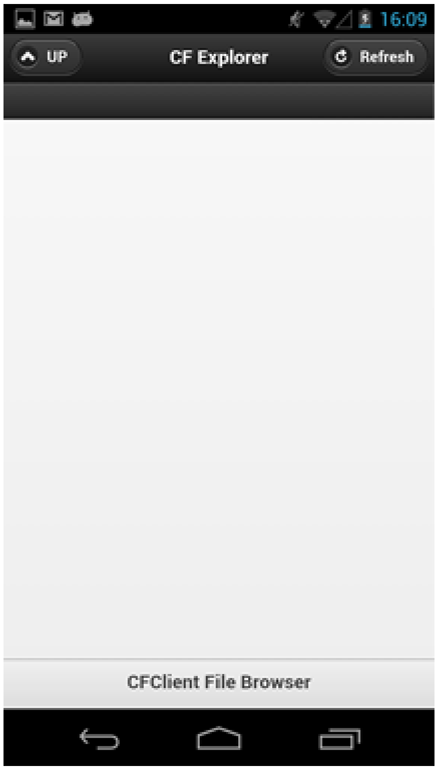
Type URL : http://10.192.28.243:8500/ExpenseTracker

http://10.192.28.243:8500/ExpenseTr

Hide Navigation Bar

WARNING : If you turn off navigation bar, then you will not be able to come back to this page. You will have to kill the application to get back to this page.

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



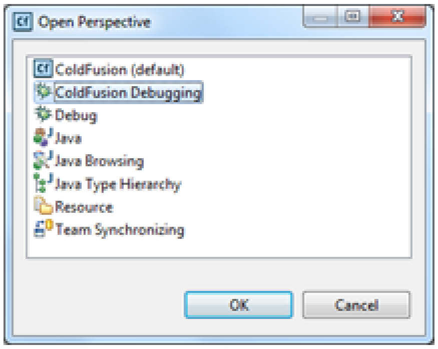
4. At this moment, you can see the control at the breakpoint position in ColdFusion Builder. Continue stepping over and debug your application.

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.

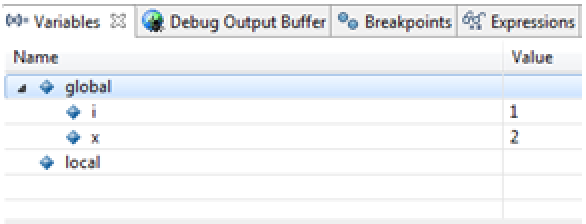
Viewing variables and evaluating expressions

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

1. Click Window > Open Perspective > Other.. and select ColdFusion Debugging:

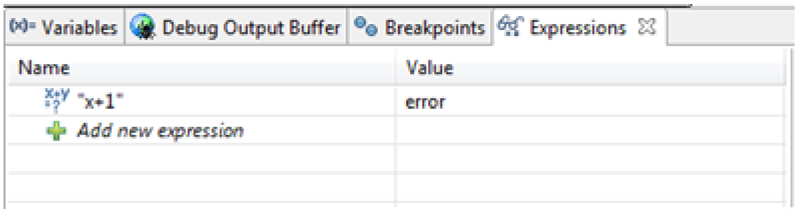


2. Click OK to continue
3. You can start viewing the variables through the following window:



Name	Value
global	
i	1
x	2
local	

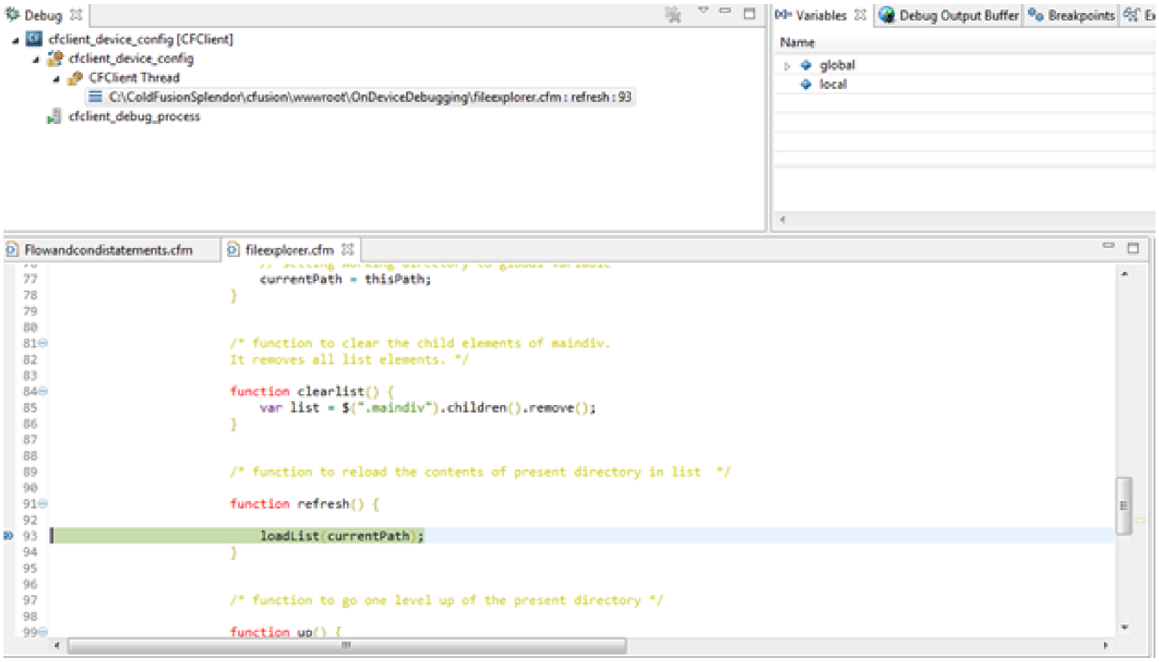
4. Similarly, you can evaluate expressions:



Name	Value
"x+1"	error
Add new expression	

Stepping in and out

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



cfclient_device_config [CFClient]
cfclient_device_config
CFClient Thread
C:\ColdFusionSplendor\cfusion\wwwroot\OnDeviceDebugging\fileexplorer.cfm : refresh : 93
cfclient_debug_process

```
            currentPath = thisPath;
        }


        /* function to clear the child elements of maindiv.
        It removes all list elements. */

        function clearlist() {
            var list = $(".maindiv").children().remove();
        }


        /* function to reload the contents of present directory in list  */

        function refresh() {

            loadList(currentPath);
        }


        /* function to go one level up of the present directory */

        function up() {
```

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

Troubleshooting tip

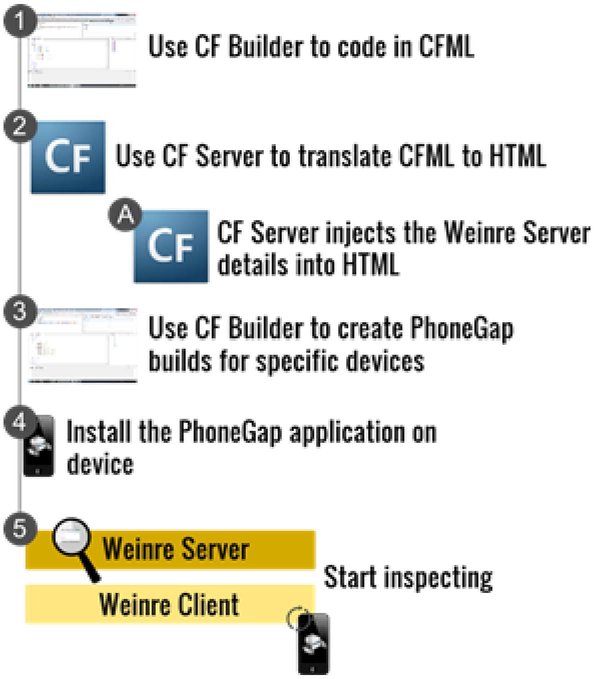
Note that on Linux, if the debug agent port is not open by default, you should manually go and enable traffic on that port through your Linux tools/firewalls (iptables). For more information on opening up ports on your machine, read the manual for your Linux distribution.
For more information on using the debugger user interface in ColdFusion Builder, see the Eclipse help.

Community videos

Inspecting Mobile Applications

Weinre (Web Inspector Remote) is a remote inspector that can be used to debug your HTML-based mobile applications generated by ColdFusion. If you have already used PhoneGap for building applications and Weinre for debugging applications, you will find the Weinre support in ColdFusion very easy to use.

The following illustration depicts how you can inspect and control your mobile applications through the Weinre client:


1
Use CF Builder to code in CFML
2
CF
Use CF Server to translate CFML to HTML
A
CF
CF Server injects the Weinre Server details into HTML
3
Use CF Builder to create PhoneGap builds for specific devices
4
Install the PhoneGap application on device
5
Weinre Server
Weinre Client
Start inspecting

Let us assume that you are packaging your ColdFusion-based mobile application into a platform-specific build targeting a particular device. You need to follow these broader steps to successfully inspect your application installed on the device:

1. Use ColdFusion Builder to develop the application. See Building mobile applications.
2. Start the local Weinre Server from the ColdFusion Server Administration Console. See Step 1 – Start the local Weinre Server. You will get the Weinre client URL when the Weinre Server starts.
3. Configure ColdFusion Builder with Weinre Server IP address and port details. See Step 3 – Configure ColdFusion Builder.
4. When you create a PhoneGap application:
    a. The ColdFusion Server translates the CFML files to equivalent HTML and JavaScript files.
    b. The ColdFusion Server also injects code into the HTML files allowing the HTML files to be inspected remotely through the Weinre Client running on a browser.
    c. Copy and install the PhoneGap application on the device.
    d. Open the application on device.
    e. From a remote browser, access the Weinre client through the URL obtained from step 2.
    f. Start inspecting the application from the browser.

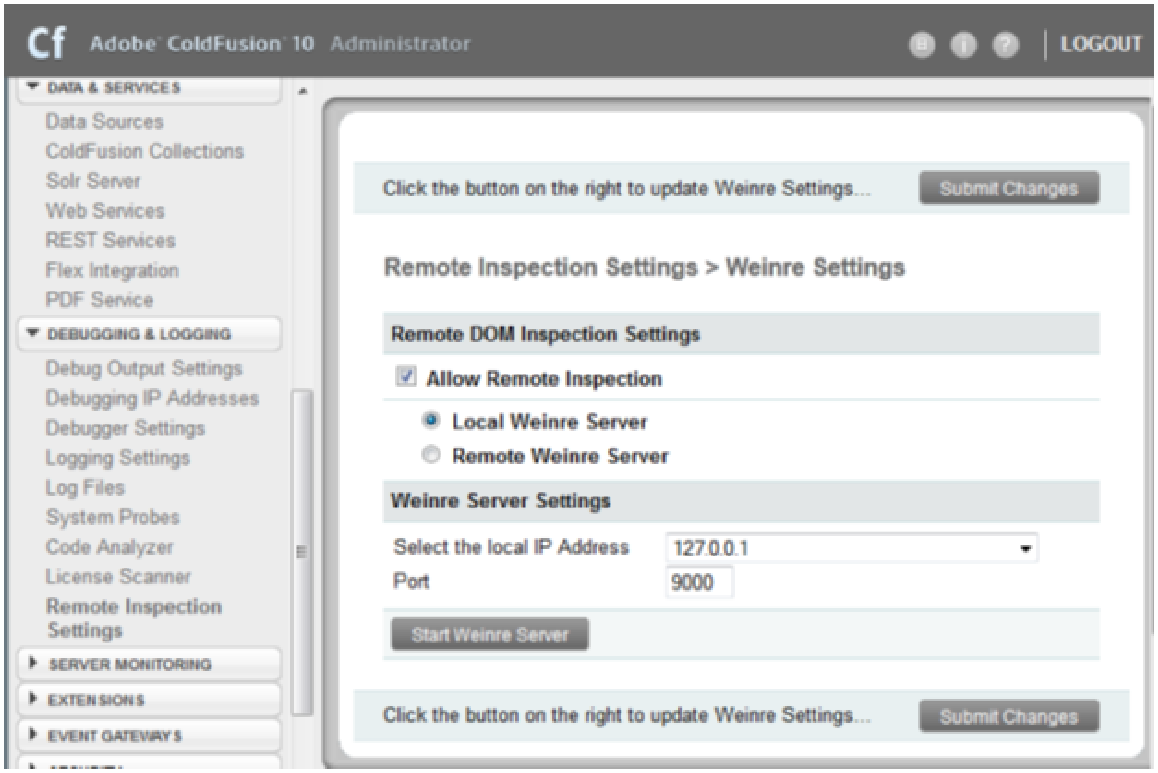
The following section describes these steps in detail.

> **Note:** A Weinre inspection server is bundled with the ColdFusion Server. All that you need to do is start the local Weinre Server and start remotely inspecting your applications installed on devices.

Step 1 – Start the local Weinre Server

To start the local Weinre Server, perform the following tasks:

1. In the ColdFusion Administration Console, click **Debugging & Logging** > **Remote Inspection Settings**
2. On the right panel, click **Allow Remote Inspection**
3. Click **Local Weinre Server**
4. Leave the default values for the IP address and port
5. Click **Start Weinre Server**



Click the button on the right to update Weinre Settings...

Remote Inspection Settings > Weinre Settings

Remote DOM Inspection Settings

☑ Allow Remote Inspection

● Local Weinre Server

○ Remote Weinre Server

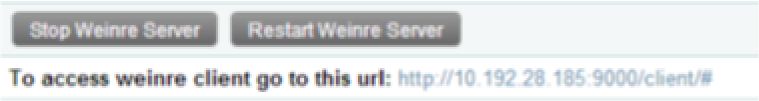
Weinre Server Settings

Select the local IP Address: 127.0.0.1

Port: 9000

Click the button on the right to update Weinre Settings...

6. When the Weinre Server gets successfully started, you can see a message containing the Weinre client URL:



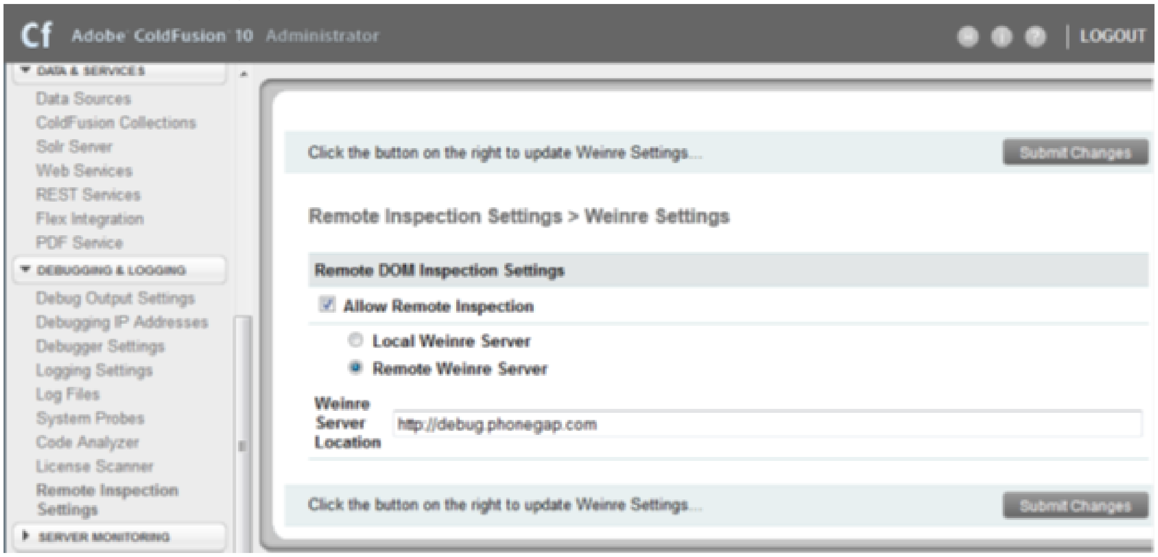
To access weinre client go to this url: http://10.192.28.185:9000/client/#

Step 2 – (Optional, if not Step 1) Provide the remote Weinre Server location

To configure a remote Weinre Server, perform the following tasks:

1. In the ColdFusion Administration Console, click **Debugging & Logging** > **Remote Inspection Settings**
2. On the right panel, click **Allow Remote Inspection**
3. Click **Remote Weinre Server**

4. Enter the Weinre server URL
5. Click **Submit Changes**



Click the button on the right to update Weinre Settings...

Remote Inspection Settings > Weinre Settings

Remote DOM Inspection Settings

☑ Allow Remote Inspection

○ Local Weinre Server

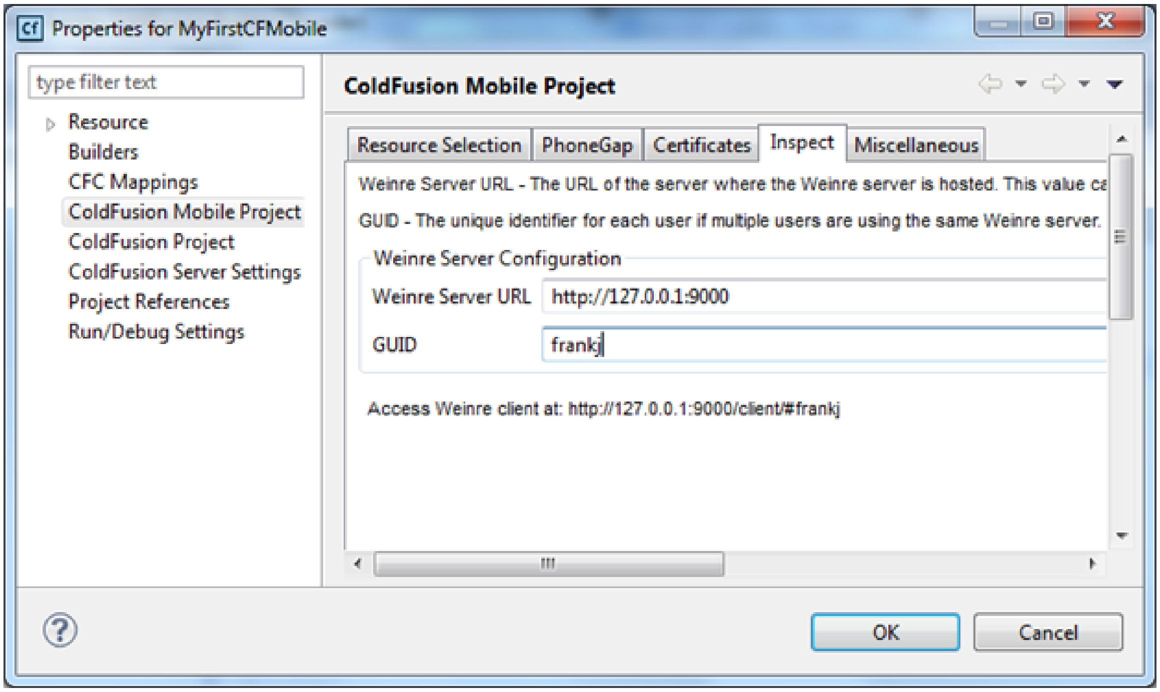
● Remote Weinre Server

Weinre Server Location: http://debug.phonegap.com

Click the button on the right to update Weinre Settings...

Step 3 – Configure ColdFusion Builder

When you have a local Weinre Server running, right-click the ColdFusion Mobile project, select Properties > ColdFusion Mobile Project and click the **Inspect** tab. Enter the Weinre Server URL. If a local Weinre Server is running provide the loopback address. You can provide a unique ID/GUID that will identify your debug sessions in the Weinre Server.



Weinre Server URL - The URL of the server where the Weinre server is hosted. This value ca

GUID - The unique identifier for each user if multiple users are using the same Weinre server.

Weinre Server Configuration

Weinre Server URL: http://127.0.0.1:9000

GUID: frankj

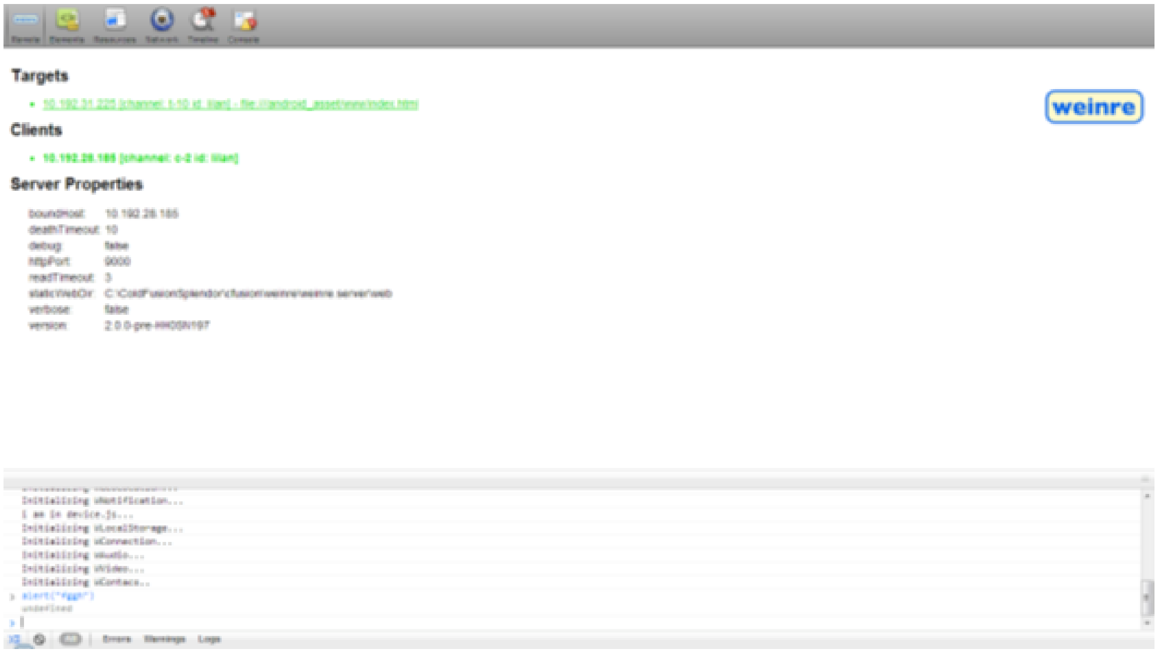
Access Weinre client at: http://127.0.0.1:9000/client/#frankj

Step 4 – Packaging the mobile application

See this section.

Step 5 – Inspecting the mobile application

You need to connect the mobile device to the same network as your Weinre Server. When the application is running on your device, open a remote browser and type the URL of the Weinre client that you have obtained in Step 1.



Targets

- 10.192.31.225 [channel: t-10 id: lilan] - file:///android_asset/www/index.html

Clients

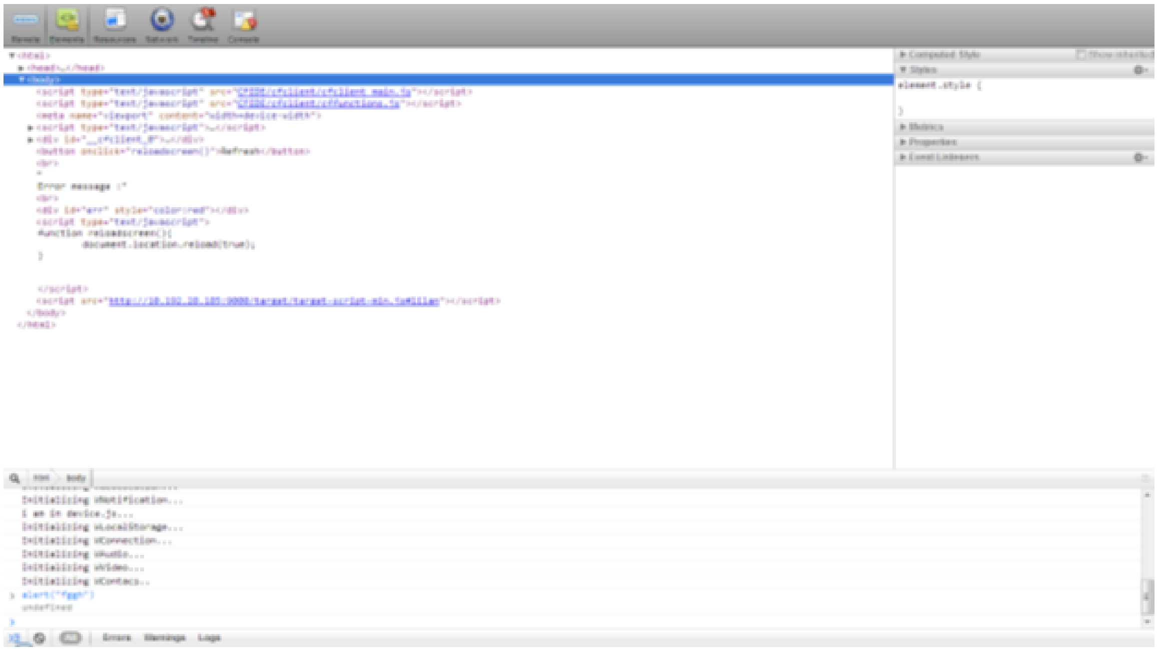
- 10.192.28.185 [channel: c-2 id: lilan]

Server Properties

```
boundHost     10.192.28.185
deathTimeout  10
debug         false
httpPort      9000
readTimeout   5
staticWebDir  C:\ColdFusionSplendor\cfusion\weinre\weinre.server\web
verbose       false
version       2.0.0-pre-HH0SN197
```

```
  Initializing cfNotification...
  I am in device.js...
  Initializing cfLocalStorage...
  Initializing cfConnection...
  Initializing cfAudio...
  Initializing cfVideo...
  Initializing cfContacts..
> alert("fggh")
  undefined
>
```

You will see the Weinre client interface. The **Targets** section lists the applications running on all connected devices available for inspection. When the hyperlink is green, the application is ready for inspection. Click the link to see the source code of the running application:



```
▼<html>
 ▶<head>…</head>
 ▼<body>
    <script type="text/javascript" src="CFIDE/cfclient/cfclient_main.js"></script>
    <script type="text/javascript" src="CFIDE/cfclient/cffunctions.js"></script>
    <meta name="viewport" content="width=device-width">
   ▶<script type="text/javascript">…</script>
   ▶<div id="__cfclient_0">…</div>
    <button onclick="reloadscreen()">Refresh</button>
    <br>
    "
    Error message :"
    <br>
    <div id="err" style="color:red"></div>
    <script type="text/javascript">
    function reloadscreen(){
            document.location.reload(true);
    }


    </script>
    <script src="http://10.192.28.185:9000/target/target-script-min.js#lilan"></script>
  </body>
</html>
```

```
element.style {
}
```

```
  Initializing cfNotification...
  I am in device.js...
  Initializing cfLocalStorage...
  Initializing cfConnection...
  Initializing cfAudio...
  Initializing cfVideo...
  Initializing cfContacts..
> alert("fggh")
  undefined
>
```

Start inspecting your application. For more information on the Weinre user interface, see this web page.

Inspecting web-based mobile application

For a web-based mobile application (not a packaged PhoneGap build), the steps mentioned above are applicable except for a few minor changes. The mobile will request for a web page with a query parameter of INSPECT in the URL. You can manually provide the URL parameters as follows:

- http://<*ip address*>:<*port*>/<file path>?INSPECT
- http:// <*ip address*>:<*port*>/<file path>?INSPECT&GUID=weinre
- http:// <*ip address*>:<*port*>/<file path>?INSPECT=false

> ℹ Note that in the case of static files, Weinre inspection will work only if these files are served by the ColdFusion Server. If the static files are served from any other server like Apache or IIS server, the inspection may not work.

Packaging Mobile Applications

Overview

Once you are done building the mobile application, you can package the application targeting some common mobile platforms. ColdFusion builder packaged applications can access the native capabilities of the mobile platform. When you are building the applications, you will be writing only CFML code and not any device-specific native code.

Ideally, you will be using ColdFusion Builder for building the mobile application and ColdFusion Server for translating the ColdFusion code in your application to corresponding HTML/JavaScript code that can be packaged and installed on the device. ColdFusion builder gets this translation done through ColdFusion Server seamlessly with a few easy configuration settings. ColdFusion Builder helps in creating platform-specific installers (.apk and .ipa) by invoking the PhoneGap build service. However, you do not need to package the application targeting individual platforms if you are not using any hardware or device-specific functionalities.

See Types of mobile applications for all the supported types while building ColdFusion-based mobile applications. Information available in this chapter is only applicable for Type 1 and Type 3 deployments.

Supported mobile platforms

The ColdFusion builder currently supports packaging applications for the following mobile platforms:

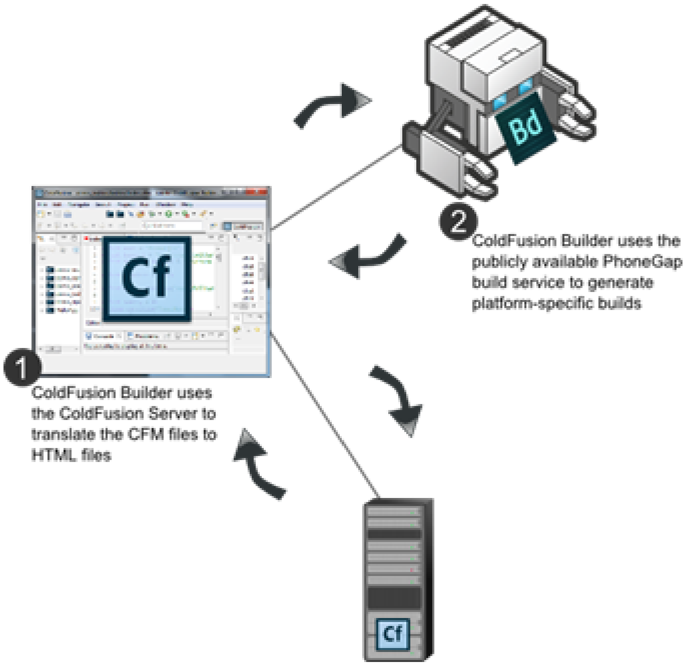
- Android 4.x or higher
- iOS 6.0 or higher

Note that the platform requirements are enforced by the PhoneGap build service and not by ColdFusion Builder.

Packaging applications using ColdFusion Builder

After creating your mobile application in ColdFusion Builder, you can generate a platform-specific package that can be installed on the mobile device (iOS and Android). ColdFusion Builder sends the ColdFusion (.cfm) files to the

ColdFusion Server, which converts the .cfm files to .html and .js files. All these files along with a generated PhoneGap configuration file (config.xml) are bundled and submitted to the Adobe's PhoneGap build service. When the PhoneGap build service completes the build process, you can download the builds to the local file system.


2
ColdFusion Builder uses the publicly available PhoneGap build service to generate platform-specific builds
1
ColdFusion Builder uses the ColdFusion Server to translate the CFM files to HTML files
Bd
Cf
Cf

Global configuration requirements

The following sections describe the global configurations required to prepare ColdFusion Builder for creating platform-specific mobile applications.

Step 1 – Get the required certificates

The ColdFusion Builder supports creating platform-specific builds for Android and iOS platforms. In order to package the mobile applications for these platforms, you need to configure the ColdFusion Builder to sign the applications with an appropriate developer/self-signed certificate. In the case of Android, providing the certificate details is optional as you can create an Android Application Package (APK) file for testing on your devices without signing it. However, testing the mobile application on iOS devices require you to have a developer certificate and a provisioning profile file.

Ensure that you follow the steps provided in this article to get started:

For iOS development

- Create and download development provisioning profiles. Note that you need to first join the iOS developer program to generate developer certificate for testing your mobile applications.

For Android development

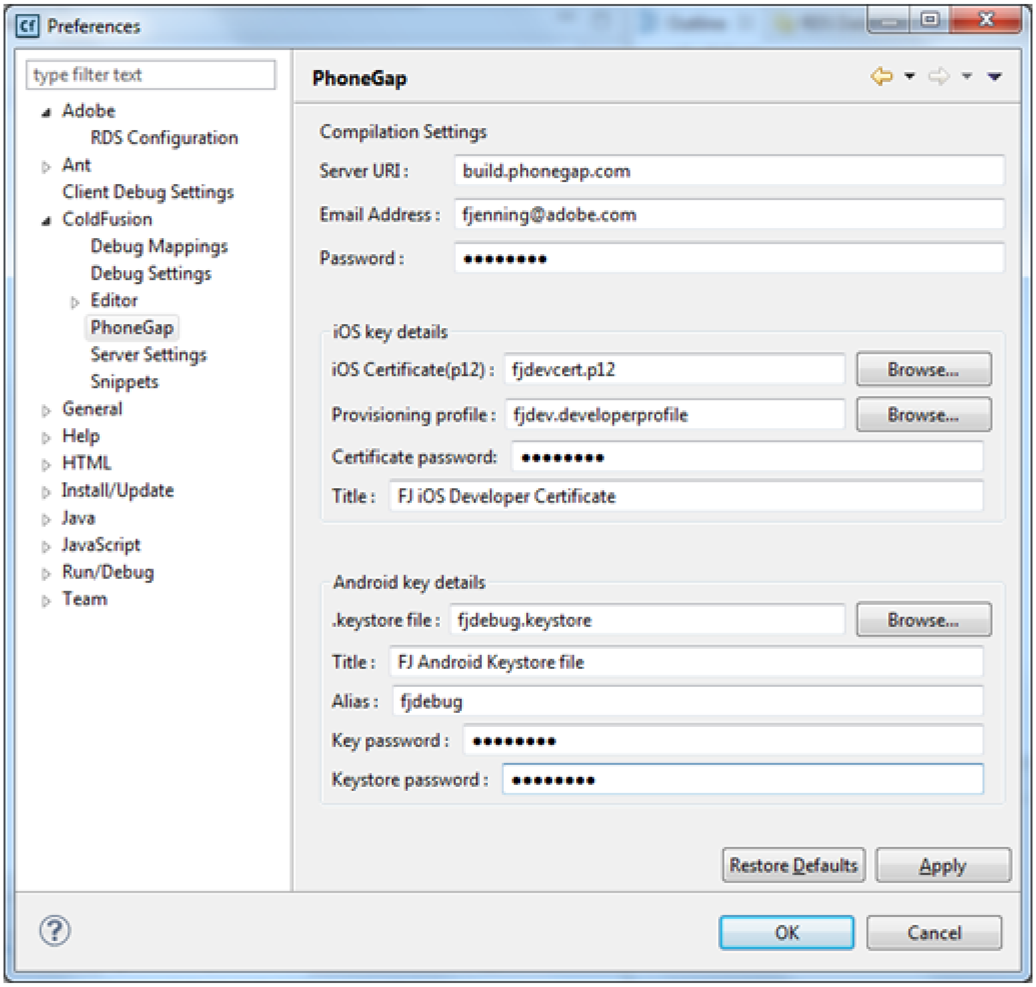
- Create the keystore file for signing applications.

Step 2 – Register for a PhoneGap plan

You need to register on the PhoneGap build service site and get a suitable PhoneGap build plan. Choose from the a vailable plans here and complete the registration. You can either register and create a new account or you can login using your Adobe ID.

Step 3 – Provide the server and authentication details

Once you have created and stored the required certificates, go to **Windows > Preferences > ColdFusion > PhoneGap** and provide the required details as shown in the following screen:



Compilation Settings

Server URI : build.phonegap.com

Email Address : fjenning@adobe.com

Password : ●●●●●●●●●

iOS key details

iOS Certificate(p12) : fjdevcert.p12

Provisioning profile : fjdev.developerprofile

Certificate password: ●●●●●●●●●

Title : FJ iOS Developer Certificate

Android key details

.keystore file : fjdebug.keystore

Title : FJ Android Keystore file

Alias : fjdebug

Key password : ●●●●●●●●●

Keystore password : ●●●●●●●●●

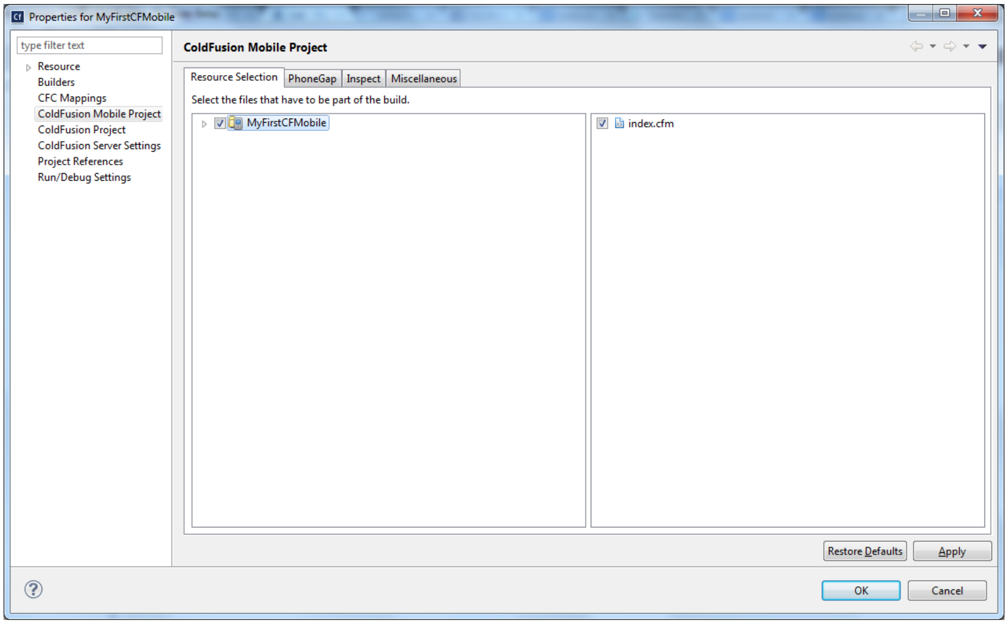
> **Important**
> **Note:** If any of the iOS key details are not specified, packaging will not work. Ensure that you specify ALL the details.

Project-specific configuration requirements

The following sections describe the project-specific configurations required to prepare ColdFusion Builder for creating platform-specific mobile applications.

Step 1 Configuring the mobile project properties

1. If you have already created a ColdFusion Builder Mobile project (see Building Mobile Applications), right-click the project in the **Navigator** panel and click **Properties**.
2. Select the **ColdFusion Mobile Project** in the left pane to see the available properties for configuration. **Note:** For packaging Server CFCs, go to the Miscellaneous tab and provide the application base URL.
3. In the **ColdFusion Mobile Project** panel, select the **Resource Selection** tab and ensure that only the CFM files and other supporting assets (under the Server's web root directory) are selected:



Select the files that have to be part of the build.

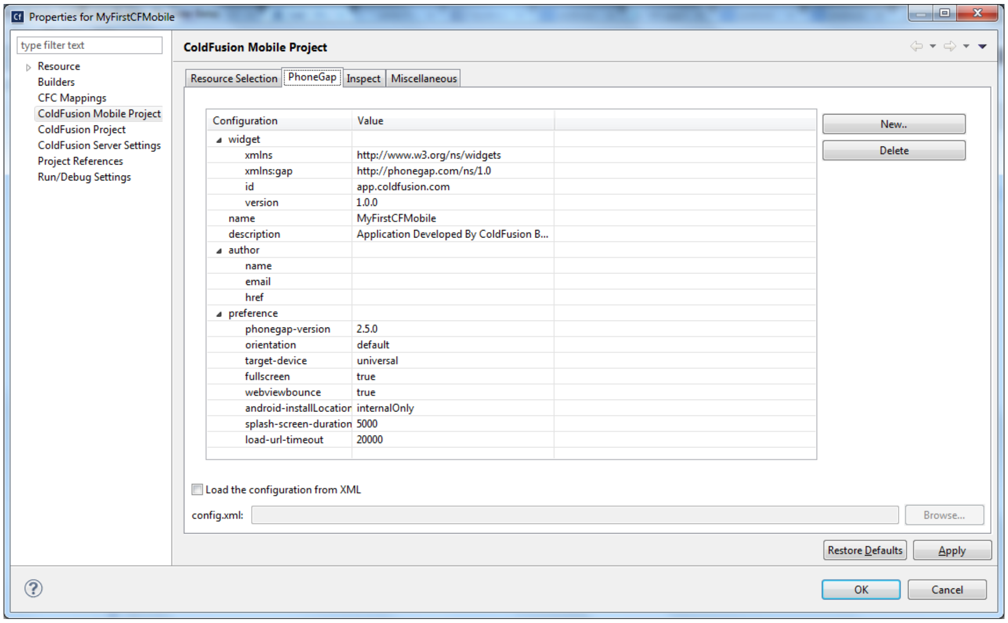
☑ MyFirstCFMobile

☑ index.cfm

Keep your CFM files and other supporting assets in a separate directory under the web root directory so that you can select just that directory.

Important: All the selected files must be present under the Server's web root directory or under web root's sub-directories. Also, it is mandatory to have an **index.cfm** file in your application. **PhoneGap mandates its index file to be present in the root folder, because of which ColdFusion has to mandate all the mobile applications developed by ColdFusion Builder should have the index.cfm or index.html in the application root folder.**

4. Select the PhoneGap tab.



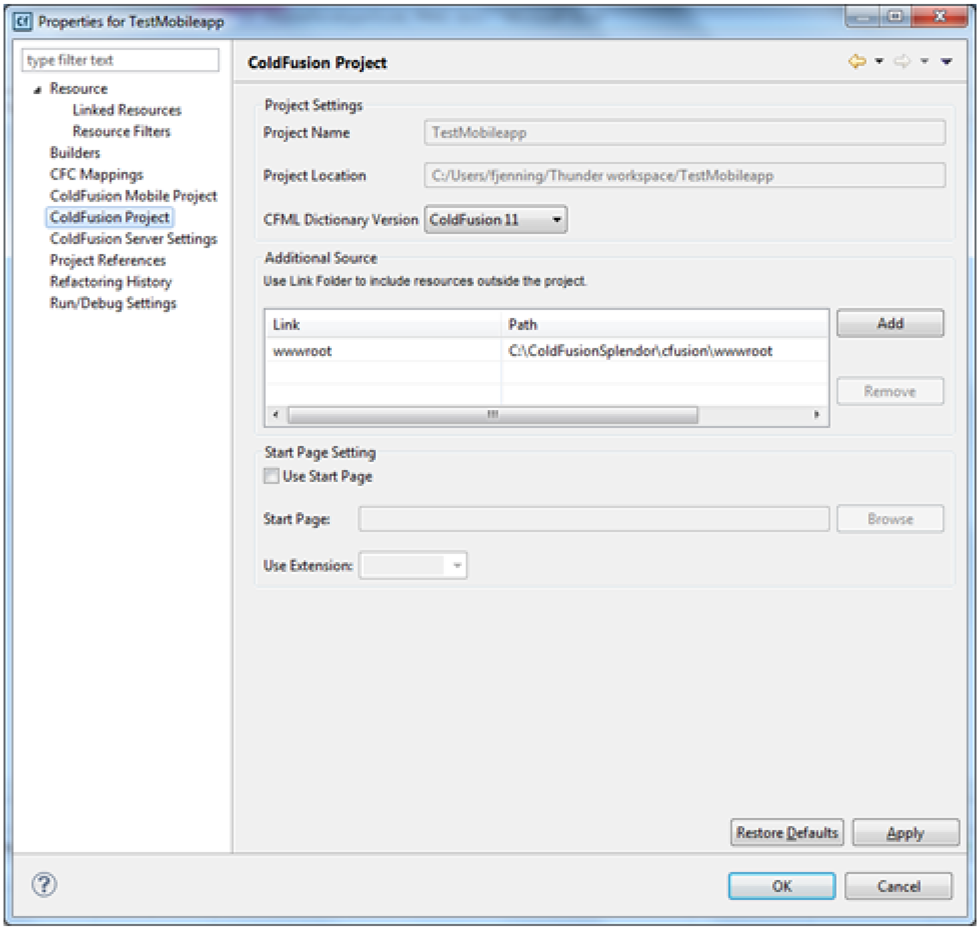
Configuration	Value
widget	
xmlns	http://www.w3.org/ns/widgets
xmlns:gap	http://phonegap.com/ns/1.0
id	app.coldfusion.com
version	1.0.0
name	MyFirstCFMobile
description	Application Developed By ColdFusion B...
author	
name	
email	
href	
preference	
phonegap-version	2.5.0
orientation	default
target-device	universal
fullscreen	true
webviewbounce	true
android-installLocatior	internalOnly
splash-screen-duration	5000
load-url-timeout	20000

□ Load the configuration from XML

config.xml:

All these properties are automatically populated by ColdFusion Builder. You just need to change the author-specific information. Alternatively, you can also load the PhoneGap settings from an external PhoneGap configuration file. See https://build.phonegap.com/docs/config-xml for a sample PhoneGap configuration. If you need to add any other platform-specific attributes, you can add them through by clicking the **Add Attribute…** button. Note that there major changes from PhoneGap version 2.9 to 3.x. If you want to manually specify a PhoneGap version, you need to understand the changes mentioned in this document.

5. Also, you need to link the web root path to your project. Click **ColdFusion Project** > **Add** to map the ColdFusion Server web root directory to a linked alias:



Project Settings

Project Name: TestMobileapp

Project Location: C:/Users/fjenning/Thunder workspace/TestMobileapp

CFML Dictionary Version: ColdFusion 11

Additional Source

Use Link Folder to include resources outside the project.

Link	Path
wwwroot	C:\ColdFusionSplendor\cfusion\wwwroot

Start Page Setting

□ Use Start Page

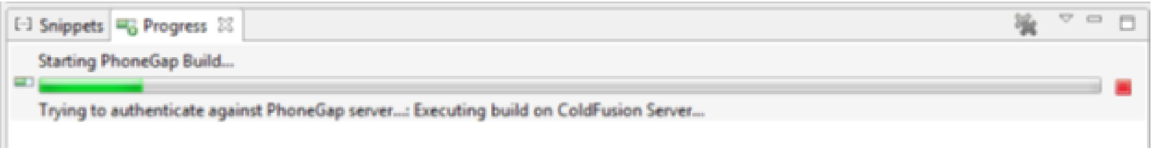
Start Page:

Use Extension:

Step 2 Invoking the PhoneGap build service

After you have configured the mobile project with PhoneGap build service settings, you can invoke the PhoneGap build service, by right-clicking the mobile project and by clicking **Generate PhoneGap Build.** You may need to wait for a few seconds to obtain the platform builds from the PhoneGap service.

If you have triggered multiple builds, you can quickly check the status of your builds by invoking the **PhoneGap build status** view by clicking **Window** > **Show View** > **Other** > **ColdFusion** > **PhoneGap Status**.



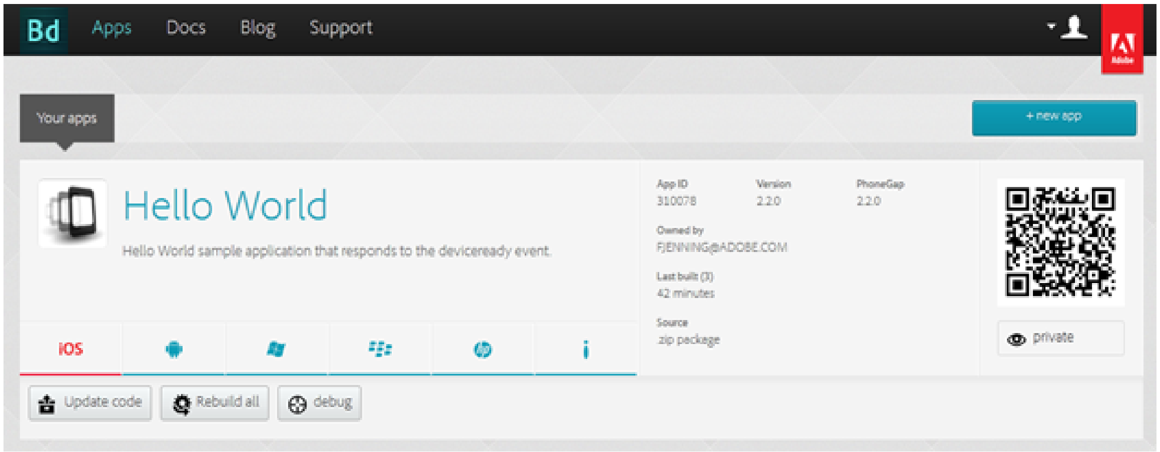
Starting PhoneGap Build...

Trying to authenticate against PhoneGap server...: Executing build on ColdFusion Server...

Note that you need to add the PhoneGap Status view to the ColdFusion perspective.

You can also log into the PhoneGap build service site and check the progress of your builds:

1. Go to https://build.phonegap.com/
2. Click **Sign in**
3. Click **Apps** and check the progress of the builds



Hello World

Hello World sample application that responds to the deviceready event.

App ID 310078

Version 2.2.0

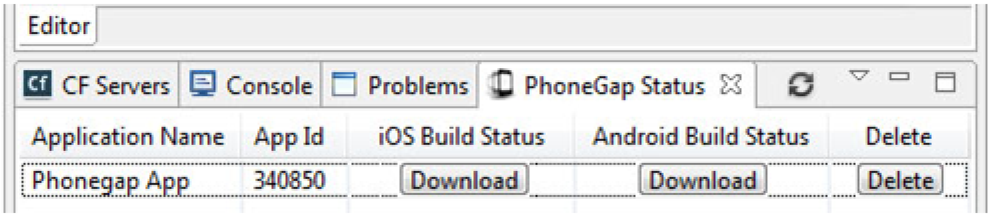
PhoneGap 2.2.0

Owned by FJENNING@ADOBE.COM

Last built (3) 42 minutes

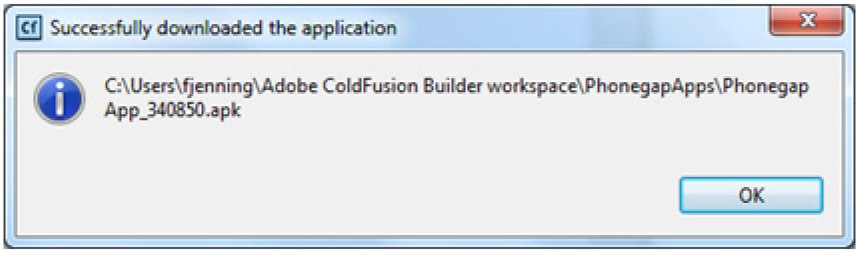
Source .zip package

If there are no errors while generating the platform-specific builds, you will see the download buttons for downloading the builds from the PhoneGap build server.



Application Name	App Id	iOS Build Status	Android Build Status	Delete
Phonegap App	340850	Download	Download	Delete

When you click the Download button, the build will get downloaded and stored in the **PhonegapApps** directory under the ColdFusion Builder workspace and you will get a notification message as shown in the following figure:



Successfully downloaded the application

C:\Users\fjenning\Adobe ColdFusion Builder workspace\PhonegapApps\Phonegap App_340850.apk

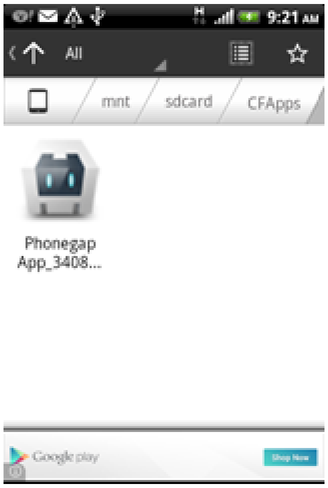
Testing the mobile application

Now that you have generated the platform-specific builds, you can test them on your mobile device to ensure that the application is functioning properly.

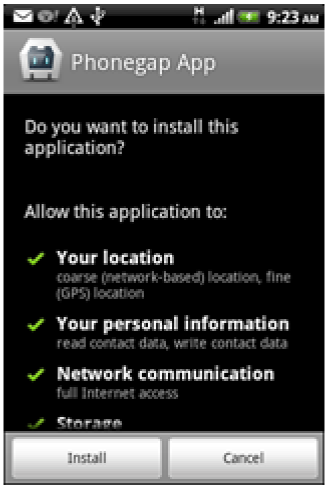
On Android

To test the application on an Android device, perform the following tasks:

1. Transfer the APK file to the SD card or the phone memory.



2. Execute the APK file to start the installation



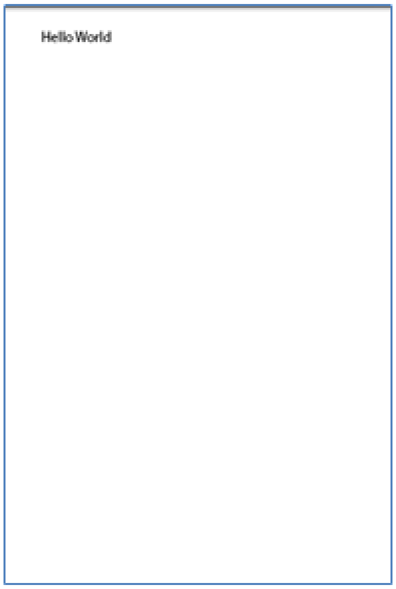
Phonegap App

Do you want to install this application?

Allow this application to:

- Your location
  coarse (network-based) location, fine (GPS) location
- Your personal information
  read contact data, write contact data
- Network communication
  full Internet access
- Storage

3. Once installed, invoke the application from the Applications menu:



Hello World

Note that you need to enable your Android device to install non-market applications. See the troubleshooting section on installing non-market applications.

On iOS

When you have an IPA file stored locally, you can use Apple iTunes software to install the application on your iOS device.

To install the test application on your device, perform the following tasks:

1. In the Finder, drag the provisioning profile (the file with the `.mobileprovision` extension) to the iTunes icon in the Dock.
2. Double-click the app archive `<App_Name>.ipa`. The app appears in the iTunes Applications list.
3. Sync your device. If the version of iOS on your device is earlier than the test application can run on, you need to update your device with the current version of iOS.

See this video to understand how IPA files can be installed on devices using Apple iTunes. Ensure that the test device is provisioned before installing the IPA file. See this document to understand device provisioning.

Loading Hybrid applications (Type 3)

You can use the **PhoneGap Shell Application** (*available for download from the pre-release forum*) to load hybrid applications on your mobile device.

To build a hybrid application, develop a ColdFusion Mobile application in ColdFusion Builder and deploy the application on ColdFusion Server. You need to include the **/CFIDE/cfclient/useragent.cfm** file that will allow your application to detect the mobile platform and will correspondingly load the platform-specific (Android or iOS) Cordova JavaScript file at runtime.

You can then run the mobile application deployed on the ColdFusion Server by entering the URL in the PhoneGap Shell Application or from your own customized shell application.

When the hybrid application is invoked, the content gets loaded from the ColdFusion Server. Also, the platform-specific Cordova-*.js file gets served by the ColdFusion Server depending on the mobile platform.

Use the PhoneGap Shell Application while developing your ColdFusion Mobile project to quickly test the application.

References

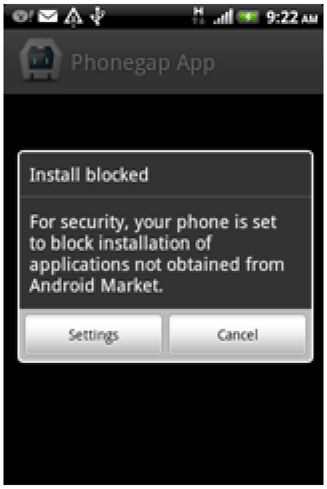
- Understand the PhoneGap build configuration

Troubleshooting Mobile Applications

The following sections discuss some of the common errors that may occur when you are developing, packaging, or deploying mobile applications.

Installing non-market applications

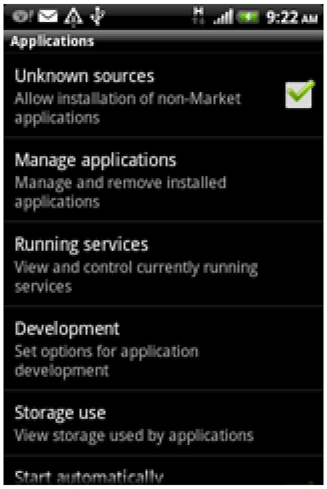
While installing your mobile application on an Android device, if you are getting the following security error, you need to enable the installation of non-market applications on your device.



Install blocked

For security, your phone is set to block installation of applications not obtained from Android Market.

Click the Setting button, and go to the appropriate screen where you can enable installing non-market applications (from unknown sources):



Unknown sources ✓
Allow installation of non-Market applications

Manage applications
Manage and remove installed applications

Running services
View and control currently running services

Development
Set options for application development

Storage use
View storage used by applications

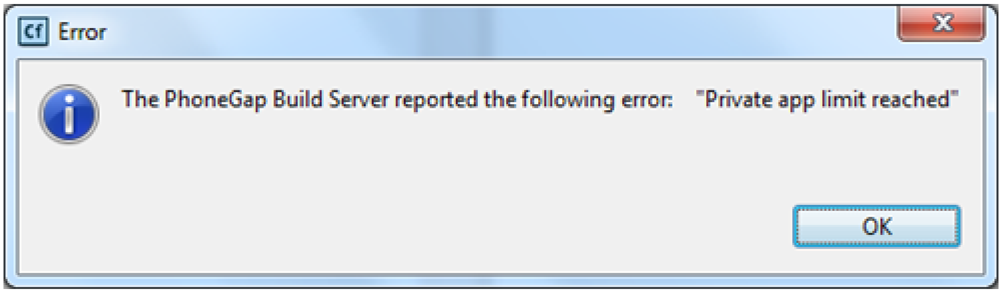
Start automatically

Note: The settings menu and the UI labels may differ depending on the version of Android.

PhoneGap application limit

In order to create new private applications using PhoneGap, you need to upgrade your subscription to a paid plan (up to 25 private applications). To upgrade your account, go to https://build.phonegap.com/plans and click on the green $9.99 paid plan button at bottom right to proceed through the registration process.

Note that the free plan allows only 1 private application. So, if you are creating more than one application, you will get the following error in ColdFusion builder:



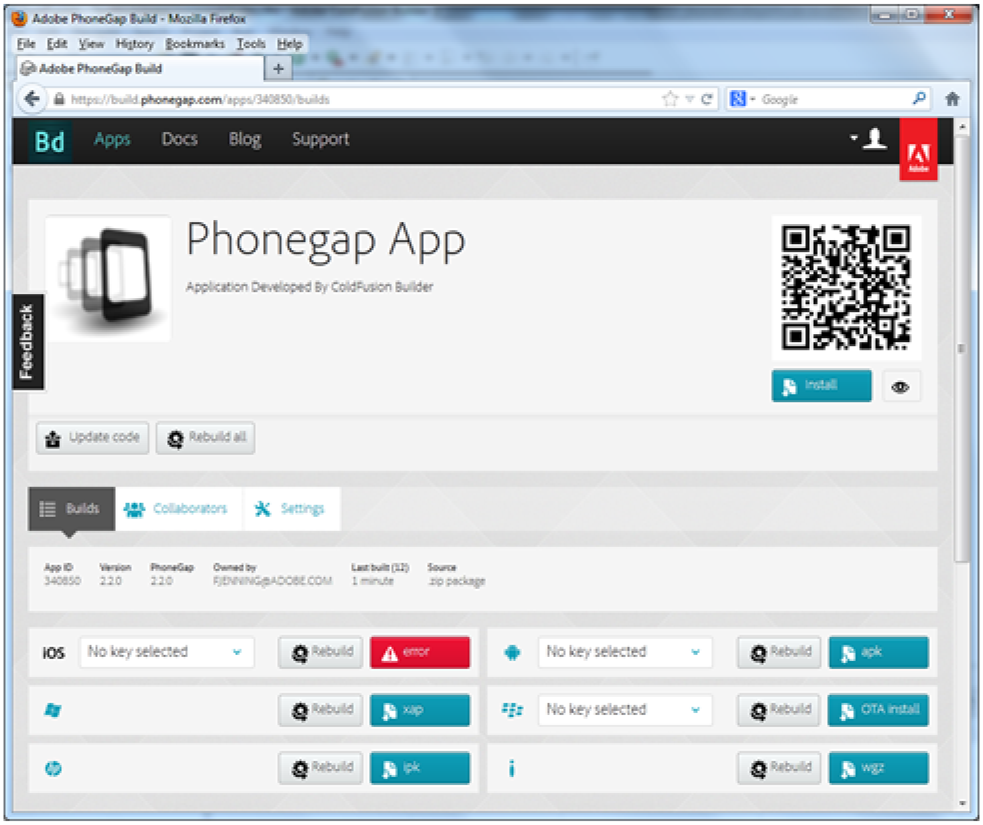
Error

The PhoneGap Build Server reported the following error: "Private app limit reached"

When this error occurs, you can either delete the existing private application or migrate to a paid PhoneGap plan.

Error while building iOS bundle

When you do not specify the right credentials for iOS packaging, PhoneGap build service will fail to produce the required IPA file:



Adobe PhoneGap Build - Mozilla Firefox

https://build.phonegap.com/apps/340850/builds

Phonegap App

Application Developed By ColdFusion Builder

App ID	Version	PhoneGap	Owned by	Last built (12)	Source
340850	2.2.0	2.2.0	FJENNING@ADOBE.COM	1 minute	.zip package

iOS: No key selected — error

No key selected — apk

No key selected — OTA install

Ensure that you have specified the path to the iOS developer certificate and the provisioning profile from the ColdFusion Builder. See Global configuration requirements.

Cannot install the IPA file

If you cannot install the IPA file on the device using Apple iTunes, check if the device is provisioned through the iOS provisioning portal. See this document for more information.

Device Detection

Detecting the device characteristics

The device detection feature of CFML allows you to identify the device properties and characteristics, which can be used to determine the best content, layout, mark-up or application to serve to the given device.

These characteristics include screen size, browser type and version, media support, and the level of support for CSS, HTML, and JavaScript.

For getting the device features and capabilities, you need to specify an attribute detectDevice in the <cfclientsettings> tag and set it to true:

```
<cfclientsettings detectDevice=true />
```

If the detectDevice attribute is set to true, ColdFusion automatically detects the features and capabilities of the device (width, height, and orientation) on which the application is running.

Note: If detectDevice is set to false, all <div> elements need to be defined before the <cfclient> block.

Supported device detection features

The following example shows the usage of the device detection feature:

```
<cfClientSettings detectDevice=true />


<cfclient>


  <cffunction access="public" name="showcanvassupport" returntype="void" >
         <cfset
document.getElementById('canvas').innerHTML=cfclient.properties.canvas>

  </cffunction>


</cfclient>


Canvas support -<b id="canvas"></b><br>
<button onclick="invokeCFClientFunction('showcanvassupport',null)">
  Show canvas support
</button>
```

In the above example, we are trying to find if the device supports HTML5 Canvas. cfclient.properties.canvas returns a boolean value indicating the support for the HTML5 Canvas property.

ColdFusion Server internally uses Modernizer JavaScript library (version 2.6.2) for the device detection feature.

The following table lists the supported device features with example usage:

Features	**Syntax**
Touch Events	cfclient.properties.touch
Canvas Text	cfclient.properties.canvastext
Canvas	cfclient.properties.canvas
Geolocation	cfclient.properties.geolocation
Web Sockets	cfclient.properties.websockets
Drag ‘n Drop	cfclient.properties.draganddrop
History	cfclient.properties.history
applicationCache	cfclient.properties.applicationcache
localStorage	cfclient.properties.localstorage
Width	cfclient.properties.width

Height	cfclient.properties.height
Device Width	cfclient.properties.deviceWidth
Device Height	cfclient.properties.deviceHeight
Orientation	cfclient.properties.orientation
Device Group Name	cfclient.properties.deviceGroupName
Device Group Descriptions	cfclient.properties.deviceGroupDescription
CSS Animations	cfclient.properties.cssanimations
CSS Columns	cfclient.properties.csscolumns
CSS Generated Content	Cfclient.properties.generatedcontent
CSS Gradients	cfclient.properties.cssgradients
CSS Reflections	cfclient.properties.cssreflections
CSS 2D Transforms	cfclient.properties.csstransforms
CSS 3D Transforms	cfclient.properties.csstransforms3d
CSS Transitions	cfclient.properties.csstransitions
Audio	cfclient.properties.audio
Video	cfclient.properties.video
Hash Change	cfclient.properties.hashchange
IndexedDB	cfclient.properties.indexeddb
Input Attributes	cfclient.properties.input.* (* refers to attributes for input elements. For possible values, see the Modernizr documentation)
Input Types	cfclient.properties.inputtypes.* (* refers to input type attributes. For possible values, see the Modernizr documentation)
Post Message	cfclient.properties.postmessage
Session Storage	cfclient.properties.sessionstorage

Web Workers	cfclient.properties.webworkers
Web SQL Database	cfclient.properties.websqldatabase

For the description on all above mentioned features, see the Modernizr documentation.

Using media queries

Media queries allow you to apply changes to the page design based on the viewing size and capability of the device on which your content is displayed. A media query consists of one or more logical expressions formed using the detected device data that checks for certain conditions of media feature and based on the result of this expression we can change the layout of the page dynamically.

If you are building a mobile application, you can easily detect the characteristics of the device and customize the layout just for that device as shown in the following example:

```
<cfclientsettings detectDevice=true />
  <cfclient>
         <cfif cfclient.properties.width lte 480 >
               <cfinclude template=" phone.css ">

               <cfelseif cfclient.properties.width gte 480 AND
cfclient.properties.width lte
                  760>
                     <cfinclude template=" tablet.css ">

               <cfelse>
               <cfinclude template=" desktop.css ">

         </cfif>
  </cfclient>
```

In the above example, the web page is customized for different devices based on their screen sizes.

Handling orientation changes

For handling the device orientation changes, you can register a listener using the addOrientationListener() function:

```
<cfclientsettings detectDevice=true />
  <cfclient>
         <cfoutput>
               Orientation : <b id="orientationId"></b><br>
               Width : <b id="width"></b><br>
               Height : <b id="height"></b><br>
         </cfoutput>
         <!--- Adding the orientation handler here. After adding
        the handler, the handler will be invoked whenever there
         is an orientation change. --->


         <cfset cfclient.addOrientationListener(orientationHandler)>



         <cffunction access="public" name="orientationHandler"
                     returntype="void" >
               <cfargument name="orientationString" type="string">
               <!--- The orientation (landscape/portrait) will be
               passed as an argument to the handler. You can also get
               the orientation value from cfclient. --->
         </cffunction>
</cfclient>
```

In the above example, addOrientationListener function is used to register a listener that monitors the orientation of the device (landscape or portrait). When the orientation of the device changes, an orientationHandler call back function is invoked.

You can use the removeOrientationListener to un-register the listener:

```
<cffunction access="public" name="removeorientationhandler"
            returntype="void" >
            <cfset cfclient.removeOrientationListener(orientationhandler)>
</cffunction>
```

You can also add multiple listeners:

```
<cfset cfclient.addOrientationListener(orientationHandler1)>
<cfset cfclient.addOrientationListener(orientationHandler2)>
```

When the device orientation changes, all the registered listener functions are invoked.

Handling window resizing events

For handling the window resizing events, you can register a listener using the addResizeListener() function:

```
<cfclientsettings detectDevice=true />
  <cfclient>
         <cfoutput>
            Width :<b id="width"></b><br>
            Height :<b id="height"></b><br>
            Device width :<b id="devicewidth"></b><br>
            Device height :<b id="deviceheight"></b><br>
         </cfoutput>
         <!--- Adding the resize handler here.
         After adding the handler, the handler will be
         invoked whenever there is a browser
         resize. --->


         <cfset cfclient.addResizeListener(resizehandler)>

         <cffunction access="public" name="resizehandler"
                     returntype="void" >
         <cfargument name="width" type="string">
         <cfargument name="height" type="string">
        <cfset document.getElementById('width').innerHTML=width>
        <cfset document.getElementById('height').innerHTML=height>
    <cfset
document.getElementById('devicewidth').innerHTML=cfclient.properties.deviceWidth>
   <cfset
document.getElementById('deviceheight').innerHTML=cfclient.properties.deviceHeight>
   </cffunction>
</cfclient>
```

You can also add multiple listeners:

```
<cfset cfclient.addResizeListener(resizeHandler1)>
<cfset cfclient.addResizeListener(resizeHandler2)>
```

When there is a change in window size all the registered resize listener functions are invoked. You can use removeResizeListener() to un-register the handlers.

```
<cffunction access="public" name="removeresizehandler"
            returntype="void" >
            <cfset cfclient.removeResizeListener(resizeHandler)>
</cffunction>
```

Setting device timeout

In the <cfclientsettings> tag, an attribute called deviceTimeOut can be specified. The default value of deviceTimeout is 10 secs. When enableDeviceApi or detectDevice is set as true, the deviceTimeOut value will be honored. Time will be provided for the required plugins to be loaded. After the specified time, an exception will be thrown.

```
<cfclientsettings detectDevice=true
deviceTimeOut="30" />
```

Client-side CFML

See Client-side CFML (for mobile development).

Mobile Templates

ColdFusion Builder allows you to create a mobile application based on pre-defined templates. These templates are basically CFML files that can use third-party web frameworks like Bootstrap or jQuery Mobile. ColdFusion Mobile Templates enhance productivity, allowing few of your developers to focus on the design of your applications, while the other developers can focus on the functionality of the applications.

ColdFusion Builder supports 2 kinds of mobile templates:

1. **System templates** - The bundled mobile templates.
2. **User templates** - The templates created by you.

Structure of mobile templates

Your mobiles templates can be in a custom directory or an archive (.zip) file. User defined templates should be stored in <cfbuilder install>/templates/mobile/user. If you want to create your own mobile template, you need to have the following mandatory items in your template directory:

- **config.xm**l - The configuration file for your template.
- **<your_template_dir> -** The directory containing the template files.

Note the following requirements:

1. You can provide a template name in the config.xml file as follows:

   ```
   <template name="My Template">
   ```

   If you do not provide a template name, the folder name or the archive (.zip) file name will be used.
2. Description can be provided in the configuration file:

   ```
   <description>
    The template description
   </description>
   ```
3. You MUST specify one or more types in your template. Types are logical grouping of your templates. For instance, all jQuery Mobile based templates can be named as 'jQueryTemplates'. In this case, you need to have a corresponding directory, 'jQueryTemplates' at teh same level as config.xml file.

An example configuration file

The following example shows a valid configuration file, config.xml:

```
<template name="My Template">
 <description>
  The template description
 </description>
 <types>
  <type value="jQuery" >
   <description>
    My mobile template containing jQuery Mobile files.
   </description>
  </type>
  <type value="javascript" />
  <type value="bootstrap">
   <description>
    My mobile template containing Bootstrap files.
   </description>
  </type>
 </types>
</template>
```

For using the mobile templates, see Building Mobile Applications.

Getting Started Examples

A simple example

Building mobile applications using the ColdFusion Mobile Platform is easy. Your code will be as simple as:

```
<cfclient>
  <cfset myvar = "Hello World">
  <cfoutput>#myvar#</cfoutput>
 </cfclient>
```

Note the new ColdFusion tag, <cfclient> that will be used for mobile development (see The new <cfclient> tag). The ColdFusion content available in the <cfclient> block gets translated into corresponding HTML/JavaScript content.

A more complex example

Let us launch the device camera:

```
<cfclient>
 <cfscript>

  function launchCamera()
  {
   // Capture the image from the device comera
   var opt = cfclient.camera.getOptions();
   var resp = cfclient.camera.getPicture(opt);
   var fileContent = cfclient.file.readFileURIAsBase64(resp);
   // Process the image
   return;
  }

 </cfscript>
</cfclient>
```

Examples

- Using the Accelerometer APIs
- Using the Audio capture APIs
- Using the Camera APIs
- Using the Connection APIs
- Using the Contact APIs
- Using the Event API
- Using the File APIs
- Using the Geolocation APIs
- Using the Storage APIs
- Using the Video capture APIs

Community posts and tutorials

- Creating database mobile application with ColdFusion Splendor
- Simplify Mobile Application Development Using ColdFusion
- Using Geolocation APIs in ColdFusion Splendor
- Taking picture and uploading to ColdFusion server
- Accessing remote data from mobile applications

Using the Accelerometer APIs

Before you begin, see Accelerometer Functions.

Setting Accelerometer options

```
<cfclientsettings enableDeviceAPI=true>
<cfclient>

 <cfset res = cfclient.accelerometer.getOptions()>
 <cfset res.frequency=20000>
 <cfset cfclient.accelerometer.setOptions(res)>

</cfclient>
```

Watching Accelerometer changes

```
<cfclientsettings enableDeviceAPI=true>
<cfclient>

 <cfset opt=cfclient.accelerometer.getOptions()>
 <!--- Assuming a <div> element is defined with id "result"--->
 <cfset document.getElementById('result').innerHTML="Options object:
#JSON.stringify(opt)#">
 <cfset watchId=cfclient.accelerometer.watch("callbackfunc",opt)>
 <cfset
document.getElementById('result').innerHTML=document.getElementById('result').innerH
TML+"Watch Id: #watchId#">

 <cffunction access="public" name="callbackfunc" returntype="void" >
  <cfargument name="acceleration">
  <cfset
document.getElementById('result').innerHTML=document.getElementById('result').innerH
TML+"<br>Value from callback function">
 </cffunction>

</cfclient>

<div id="result"/>
```

Using the Audio capture APIs

Before you begin, see Audio Functions.

A simple example for audio capture

```
<cfclientsettings enableDeviceAPI=true>

<cfclient>


    <cfset  opt = cfclient.audio.getOptions()>
    <cfset  cfclient.audio.capture(opt,'func1')>


    <cffunction name="func1">
        <cfargument name="mediaFileArray">
          <cfset document.getElementById('result').innerHTML="<br>Value from
callback function">
          <cfset
document.getElementById('result').innerHTML=document.getElementById('result').innerH
TML+JSON.stringify(mediaFileArray)>
        <cfset
document.getElementById('result').innerHTML=document.getElementById('result').innerH
TML+mediaFileArray[1].fullPath>
   </cffunction>

</cfclient>
<div id="result"/>
```

Record and play audio

```
<cfclientsettings enableDeviceAPI=true>

<cfclient>

 <cfset  mediaFileArray=cfclient.audio.capture()>

 <cfset medFil=mediaFileArray[1].fullPath>


 <!--- Play the media file --->
 <cffunction name="playing">
  <cfset medObj = cfclient.audio.play(medFil)>
 </cffunction>

 <!--- Pause the media --->
 <cffunction name="pausing">
  <cfset cfclient.audio.pause(medObj)>
 </cffunction>
```

```
<!--- Release the media --->
<cffunction name="releasing">
 <cfset cfclient.audio.release(medObj)>
</cffunction>

<!--- Stop the media --->
<cffunction name="stopping">
 <cfset cfclient.audio.stop(medObj)>
</cffunction>

<!--- Get current position of the media file --->
<cffunction name="getcurpos">
 <cfset pos = cfclient.audio.getCurrentPosition(medObj)>
</cffunction>

<!--- Go to a position in the media file --->
<cffunction name="seekingto">
 <cftry>
  <cfset pos = cfclient.audio.seekTo(medObj,2)>
  <cfcatch type="any">
   <cfset alert(cfcatch.message)>
  </cfcatch>
 </cftry>
</cffunction>

<!--- Start recording --->
<cffunction name="rec">
 <cftry>
  <cfset pos = cfclient.audio.record(medObj)>
  <cfcatch type="any">
   <cfset alert(cfcatch.message)>
  </cfcatch>
 </cftry>
</cffunction>


<cffunction name="stoprec">
 <cfset cfclient.audio.stopRecording(medObj)>
 </cffunction>

</cfclient>
```

Using the Camera APIs

Before you begin, see Camera Functions.

Launching the camera and capturing images

```
<cfclientsettings enableDeviceAPI=true>

<cfclient>


<cffunction access="public" name="getPicCam" returntype="void" >
<cfset opt = cfclient.camera.getOptions()>
<cfset res = cfclient.camera.getPicture(opt,false)>
<cfset imgStr=cfclient.file.readAsBase64(res)>
<cfset document.getElementById('myimg').src=imgStr>
<cfset document.getElementById('fileName').innerHTML=res>
</cffunction>
</cfclient>
<div id="fileName"/>
<img style = "width:200; height:100;" id = "myimg"></img>
<button onclick="invokeCFClientFunction('getPicCam',null)">getPicture -
JPEG</button>
```

Getting images from different sources

```
<cfclientsettings enableDeviceAPI=true>

<cfclient>

    <!--- Getting picture from device camera --->
    <cffunction access="public" name="getPiccam" returntype="void" >
        <cfset opt = cfclient.camera.getOptions()>
        <cfset res = cfclient.camera.getPicture(opt,true)>
        <cfset res='data:image/jpeg;base64,'+res>
        <cfset document.getElementById('myimg').src=#res#>
    </cffunction>

    <!--- Getting picture from the album --->
    <cffunction access="public" name="getPicalbm" returntype="void" >
        <cfset opt = cfclient.camera.getOptions()>
        <cfset res = cfclient.camera.getPictureFromAlbum(opt,true)>
        <cfset res='data:image/jpeg;base64,'+res>
        <cfset document.getElementById('myimg').src=#res#>
    </cffunction>

    <!--- Getting picture from the photo library --->
    <cffunction access="public" name="getPiclib" returntype="void" >
        <cfset opt = cfclient.camera.getOptions()>
        <cfset res = cfclient.camera.getPictureFromPhotoLibrary(opt,true)>
        <cfset res='data:image/jpeg;base64,'+res>
        <cfset document.getElementById('myimg').src=#res#>
    </cffunction>

</cfclient>
<img style = "width:300; height:300;" id = "myimg"></img>
<button onclick="invokeCFClientFunction('getPiccam',null)">Camera</button><br>
<button onclick="invokeCFClientFunction('getPicalbm',null)">album</button><br>
<button onclick="invokeCFClientFunction('getPiclib',null)">Library</button><br>
```

Setting camera options

```
<cfclientsettings enableDeviceAPI=true>

<cfclient>

 <cffunction access="public" name="setopt" returntype="void" >
  <cfset options = {"quality" = "40", "encodingType" = "png"}>
  <cfset cfclient.camera.setOptions(options)>
 </cffunction>

 <button onclick="invokeCFClientFunction('setopt',null)">Set Camera Options</button>


</cfclient>
```

Getting camera options

```
<cfclientsettings enableDeviceAPI=true>

<cfclient>

 <cffunction access="public" name="getopt" returntype="void" >
  <cfset res = cfclient.camera.getOptions()>
 </cffunction>

 <button onclick="invokeCFClientFunction('getopt',null)">Get Camera Options</button>

</cfclient>
```

Storing the images

```
<cfclientsettings enableDeviceAPI=true>

<cfclient>
    <cftry>
      <cfset opt = cfclient.camera.getOptions()>
        <cfset res = cfclient.camera.getPicture(opt,false)>
          <cfif !cfclient.file.directoryExists('mydir')>
            <cfset cfclient.file.createDirectory('mydir')>
        </cfif>
         <cfif opt.encodingtype eq "jpeg">
            <cfset cfclient.file.copy(res,'mydir/img.jpg')>
            <cfif cfclient.file.exists('mydir/img.jpg')>
                <cfset filObj=cfclient.file.get('mydir/img.jpg')>
                <cfset
document.getElementById('result').innerHTML=#filObj.fullPath#>
            </cfif>
            <cfelse>
                <cfset cfclient.file.copy(res,'mydir/img1.png')>
                <cfif cfclient.file.exists('mydir/img1.png')>
                    <cfset filObj=cfclient.file.get('mydir/img1.png')>
                    <cfset
document.getElementById('result').innerHTML=#filObj.fullPath#>
                </cfif>
        </cfif>
<cfcatch type="any">
<cfset alert(cfcatch.message)>
</cfcatch>
        </cftry>

</cfclient>
<div id="result"/>
```

Cleaning up

```
<cfclientsettings enableDeviceAPI=true>

<cfclient>

 <cffunction access="public" name="cleanup" returntype="void" >
  <cftry>
   <cfset cfclient.camera.cleanup()>

   <cfcatch type="any">
    <cfset alert(cfcatch.message)>
   </cfcatch>
  </cftry>
 </cffunction>

<button onclick="invokeCFClientFunction('cleanup',null)">Cleanup
now</button><br><br>

</cfclient>
```

Using the Connection APIs

Before you begin, see Connection Functions.

Getting the connection type

```
<cfclientsettings enableDeviceAPI=true>

<cfclient>

 <cfset  conType = cfclient.connection.getType()>
 <cfset
document.getElementById('result').innerHTML=document.getElementById('result').innerH
TML+"Type: #conType#">

</cfclient>

<div id="result"/>
```

Valid connection types are Connection.UNKNOWN, Connection.ETHERNET, Connection.WIFI, Connection.CELL_2G, Connection.CELL_3G, Connection.CELL_4G, Connection.CELL, and Connection.NONE.

Finding if the connection is offline

```
<cfclientsettings enableDeviceAPI=true>

<cfclient>

 <cfset cfclient.connection.onOffline('callback1')>

 <cffunction name="callback1" >
  <cfset
document.getElementById('result').innerHTML=document.getElementById('result').innerH
TML+"The mobile connection has been lost">
 </cffunction>

</cfclient>

<div id="result"/>
```

Finding if the connection is online

```
<cfclientsettings enableDeviceAPI=true>

<cfclient>

 <cfset #cfclient.connection.onOnline('callback1')#>

 <cffunction name="callback1" >
  <cfset
document.getElementById('result').innerHTML=document.getElementById('result').innerHTML+"The mobile connection is active now">
 </cffunction>

</cfclient>
```

Using the Contact APIs

See the Contact Functions to understand these examples.

Creating contacts

```
<cfinclude template="/CFIDE/cfclient/useragent.cfm">
 <cfclientsettings enableDeviceAPI=true>

 <cfclient>

  <cfset res =
cfclient.contacts.create('Joe','1234567890','joe@mycompany.com',true)>
  <cfset document.getElementById('result').innerHTML="Options object:
#JSON.stringify(Object.keys(res))#">

 </cfclient>


 <div id="result"/>
```

Adding fields to the contact

```
<cfinclude template="/CFIDE/cfclient/useragent.cfm">
 <cfclientsettings enableDeviceAPI=true>

 <cfclient>

  <cfset res.addPhoneNumber("5678234","work",true)>
  <cfset res.addEmail("joe@gmail.com","home",true)>

  <cfset cntadr=cfclient.contacts.createAddress("First street", "North", "Indian",
"India","12345", "Good address form", "home", "true")>
  <cfset res.addAddress(cntadr)>

  <cfset cntorg=cfclient.contacts.createOrganization("SciTech", "Science", "Science
Technology", "Education","true")>
  <cfset res.addOrganization(cntorg)>

  <cfset res.addIm("myIMHandle","home",true)>

  <cfset res.addCategory("Friends","home",true)>
  <cfset res.addURL("www.joe.com","home",true)>
  <cfset res.addPhoto("myDir/img.jpg","home",true)>

  <cfset cfclient.contacts.save(res)>

 </cfclient>
```

Finding a contact

```
<cfclientsettings enableDeviceAPI=true>
<cfclient>

  <!--- Searching all fields --->
  <cfset cntlst=cfclient.contacts.find("Joe",["*"])>

  <cfset
document.getElementById('result').innerHTML=document.getElementById('result1').inner
HTML+"Number of contacts - #arrayLen(cntlst)#">
  <cfscript>
   for (i = 1; i <= #arrayLen(cntlst)#; i = i + 1) {
     writeOutput("<br><br>");
        writeOutput(iter(cntlst[i]));
    }
  </cfscript>

  <!--- Searching only the displayName field --->
  <cfset cntlst=cfclient.contacts.find("Joe",["name","displayName"])>

  <cfset
document.getElementById('result').innerHTML=document.getElementById('result2').inner
HTML+"Number of contacts - #arrayLen(cntlst)#">

  <cfscript>
   for (i = 1; i <= #arrayLen(cntlst)#; i = i + 1) {
     writeOutput("<br><br>");
        writeOutput(iter(cntlst[i]));
    }
  </cfscript>


  <!--- Searching in the displayName and phoneNumbers fields --->
  <cfset cntlst=cfclient.contacts.find("user1_1",["phoneNumbers,displayName"])>
  <cfset
document.getElementById('result').innerHTML=document.getElementById('result3').inner
HTML+"Number of contacts - #arrayLen(cntlst)#">
  <cfscript>
   for (i = 1; i <= #arrayLen(cntlst)#; i = i + 1) {
     writeOutput("<br><br>");
        writeOutput(iter(cntlst[i]));
    }
  </cfscript>

</cfclient>

<div id="result1"/>
<div id="result2"/>
<div id="result3"/>
```

Getting all the contacts

```
<cfclientsettings enableDeviceAPI=true>
<cfclient>

 <cfoutput>Getting all contacts</cfoutput>
 <cfset cntlst=cfclient.contacts.getAllContacts(["*"])>
 <cfset
document.getElementById('result').innerHTML=document.getElementById('result3').inner
HTML+"Number of contacts - #arrayLen(cntlst)#">
 <cfscript>
  for (i = 1; i <= #arrayLen(cntlst)#; i = i + 1) {
    writeOutput("<br><br>");
       writeOutput(iter(cntlst[i]));
   }
 </cfscript>

 <cfoutput>Getting only the phone numbers</cfoutput>
 <cfset cntlst=cfclient.contacts.getAllContacts(["phoneNumbers"])>
 <cfset
document.getElementById('result').innerHTML=document.getElementById('result3').inner
HTML+"Number of contacts - #arrayLen(cntlst)#">
 <cfscript>
  for (i = 1; i <= #arrayLen(cntlst)#; i = i + 1) {
    writeOutput("<br><br>");
       writeOutput(iter(cntlst[i]));
   }
 </cfscript>

 <cfoutput>Getting only the display names and phone numbers</cfoutput>
 <cfset cntlst=cfclient.contacts.getAllContacts(["phoneNumbers,displayName"])>
 <cfset
document.getElementById('result').innerHTML=document.getElementById('result3').inner
HTML+"Number of contacts - #arrayLen(cntlst)#">
 <cfscript>
  for (i = 1; i <= #arrayLen(cntlst)#; i = i + 1) {
    writeOutput("<br><br>");
       writeOutput(iter(cntlst[i]));
   }
 </cfscript>


</cfclient>
```

Using the Event API

Before you begin, see Event Functions

Handling the back button event

```
<cfclientsettings enableDeviceAPI=true>
<cfclient>

 <cfset cfclient.events.onBackButton("callbackfunc")>

 <cffunction access="public" name="callbackfunc" returntype="void" >
  <!--- Back button pressed --->
 </cffunction>


</cfclient>
```

Handling the critical battery event

```
<cfclientsettings enableDeviceAPI=true>
<cfclient>

 <cfset cfclient.events.onBatteryCritical("callbackfunc")>

 <cffunction access="public" name="callbackfunc" returntype="void" >
  <!--- Battery critically low! --->
 </cffunction>
</cfclient>
```

Handling the low battery event

```
<cfclientsettings enableDeviceAPI=true>
<cfclient>

 <cfset cfclient.events.onBatteryLow("callbackfunc")>

 <cffunction access="public" name="callbackfunc" returntype="void" >
  <!--- Battery low! --->
 </cffunction>
</cfclient>
```

Using the File APIs

Before you begin, read File System Functions.

Uploading a file from the device

```
<cfclientsettings enabledeviceapi="true" enabledebuglog="true">


<cfclient>

 <cffunction name="uploadSuccess" >
  <cfset alert("uploaded")>
 </cffunction>

 <cffunction name="uploadError" >
  <cfargument name="err" >
  <cfset alert(err.code)>
 </cffunction>

 <cftry>
     <cfscript>

   cfclient.file.write("a.txt","hello");

cfclient.file.upload("a.txt","http://<Your-CF-Server>:8500/fileUpload/files/upload.c
fm",uploadSuccess,uploadError);

     </cfscript>
     <cfcatch type="Any" >
      <cfset alert("exception " & cfcatch.message)>
     </cfcatch>
    </cftry>

</cfclient>
```

Downloading a file to the device

```
<cfclientsettings enabledeviceapi="true" enabledebuglog="true">
<cfclient>
 <cffunction name="uploadSuccess" >
  <cfargument name="obj" >

  <cfset alert("downloaded "+obj)>
  <cfset alert(cfclient.file.read('b.txt'))>
 </cffunction>

 <cffunction name="uploadError" >
  <cfargument name="err" >

  <cfset alert(err.code)>

 </cffunction>

 <cftry>
     <cfscript>


cfclient.file.download("http://<Your-CF-Server>:8500/fileUpload/files/b.txt","b.txt"
,uploadSuccess,
    uploadError);

     </cfscript>
     <cfcatch type="Any" >
      <cfset alert("exception " & cfcatch.message)>
     </cfcatch>
    </cftry>

</cfclient>
```

Creating a directory

```
<cfclientsettings enableDeviceAPI=true>
<cfclient>

 <cfset cfclient.file.createDirectory('MyDir')>

</cfclient>
```

Removing a directory

```
<cfclientsettings enableDeviceAPI=true>
<cfclient>

 <cfif cfclient.file.directoryExists('MyDir')>
  <cfset cfclient.file.removeDirectory('MyDir',true)>
 </cfif>

</cfclient>
```

Getting the working directory

```
<cfclientsettings enableDeviceAPI=true>
<cfclient>

 <cfset cwd = cfclient.file.getWorkingDirectory()>

</cfclient>
```

Copying a directory

```
<cfclientsettings enableDeviceAPI=true>
<cfclient>

 <cfif cfclient.file.directoryExists('dir1')>
  <cfset cfclient.file.removeDirectory('dir1',true)>
 </cfif>

 <cfset dir1Obj=cfclient.file.createDirectory('dir1')>


 <cfif cfclient.file.directoryExists('dir2')>
  <cfset cfclient.file.removeDirectory('dir2',true)>
 </cfif>


 <cfset dir2Obj=cfclient.file.createDirectory('dir2')>

 <cfset cfclient.file.copyDirectory(dir2Obj.fullPath,dir1Obj.fullPath)>


</cfclient>
```

Reading a file

```
<cfclientsettings enableDeviceAPI=true>
<cfclient>

  <cfset fileContent = cfclient.file.read('MyFile.txt')>
</cfclient>
```

Reading a file as a Base64 text

```
<cfclientsettings enableDeviceAPI=true>
<cfclient>

 <cfset opt = cfclient.camera.getOptions()>
 <cfset res = cfclient.camera.getPicture(opt,false)>
 <cfset imgStr=cfclient.file.readAsBase64(res)>

</cfclient>
```

Using the Geolocation APIs

Before you begin, see Geolocation Functions

Getting the current position

```
<cfclientsettings enableDeviceAPI=true>
<cfclient>

 <cfset res = cfclient.geolocation.getOptions()>
 <cfset curpos = cfclient.geolocation.getCurrentPosition(res)>
 <cfset
document.getElementById('result').innerHTML=document.getElementById('result1').inner
HTML+"Latitude : #curpos.coords.latitude#">
 <cfset
document.getElementById('result').innerHTML=document.getElementById('result2').inner
HTML+"Longitude : #curpos.coords.longitude#">
 <cfset
document.getElementById('result').innerHTML=document.getElementById('result3').inner
HTML+"Altitude : #curpos.coords.altitude#">
 <cfset
document.getElementById('result').innerHTML=document.getElementById('result4').inner
HTML+"Accuracy : #curpos.coords.accuracy#">
 <cfset
document.getElementById('result').innerHTML=document.getElementById('result5').inner
HTML+"Altitude Accuracy : #curpos.coords.altitudeAccuracy#">
 <cfset
document.getElementById('result').innerHTML=document.getElementById('result6').inner
HTML+"Heading : #curpos.coords.heading#">
 <cfset
document.getElementById('result').innerHTML=document.getElementById('result7').inner
HTML+"Speed : #curpos.coords.speed#">
 <cfset
document.getElementById('result').innerHTML=document.getElementById('result8').inner
HTML+"Timestamp : #curpos.timestamp#">
</cfclient>

<div id="result1"/>
<div id="result2"/>
<div id="result3"/>
<div id="result4"/>
<div id="result5"/>
<div id="result6"/>
<div id="result7"/>
<div id="result8"/>
```

Setting Geolocation options

```
<cfclientsettings enableDeviceAPI=true>
<cfclient>

 <cfset res = cfclient.geolocation.getOptions()>
 <cfset res.enablehighaccuracy=false>
 <cfset res.maximumage=3000>
 <cfset res.timeout=5000>
 <cfset cfclient.geolocation.setOptions(res)>


</cfclient>
```

Monitoring (watching) the current position

```
<cfclientsettings enableDeviceAPI=true>
<cfclient>

 <cfset res = cfclient.geolocation.getOptions()>
 <cfset res.enableHighAccuracy=false>
 <cfset res.maximumAge=1000>
 <cfset res.timeout=2000>

 <cfset watchId=cfclient.geolocation.watchPosition(callbackfunc,res)>

 <cffunction access="public" name="callbackfunc" returntype="void" >
  <cfargument name="position">
   <cfset alert(iter(position))>
 </cffunction>
</cfclient>
```

Using the Storage APIs

Before you begin, see Storage Functions.

Setting/getting an item

```
<cfclientsettings enableDeviceAPI=true>

<cfclient>

 <cfset  cfclient.localstorage.setItem("key1","value1")>
 <cfset keyVal = cfclient.localstorage.getItem("key1")>


</cfclient>
```

Removing an item

```
<cfclientsettings enableDeviceAPI=true>

<cfclient>

 <cfset  cfclient.localstorage.setItem("key1","value1")>
 <cfset keyVal = #cfclient.localstorage.getItem("key1")>

 <cfset cfclient.localstorage.removeItem("key1")>

</cfclient>
```

Using the Video capture APIs

Before you begin, see Video Functions.

Capturing the video

```
<cfclientsettings enableDeviceAPI=true>

<cfclient>

 <cfset opt = cfclient.video.getOptions()>
 <cfset  cfclient.video.capture(opt,'func1')>

 <cffunction name="func1">
  <cfargument name="mediaFileArray">
  <cfset document.getElementById('result').innerHTML="Options object:
#JSON.stringify(mediaFileArray)#">
 </cffunction>

</cfclient>

<div id="result"/>
```