

Lab 1: Getting Started with IBM Worklight – Lab Exercise



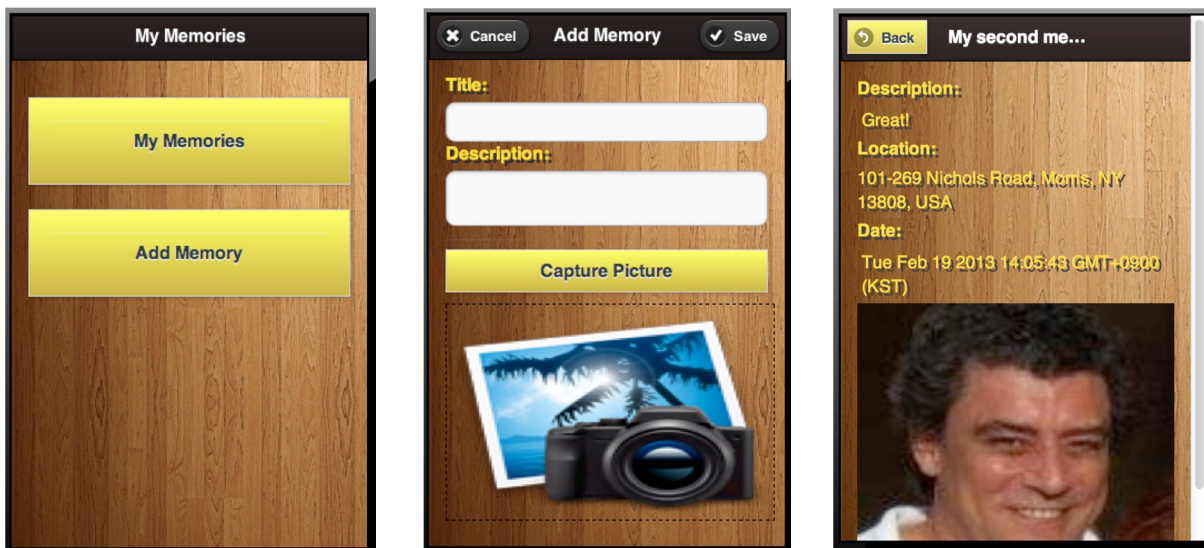
Table of Contents

1. Getting Started with IBM Worklight	3
1.1 Start Worklight Studio	5
1.1.1 Start Worklight Studio	6
1.2 Create new MyLab1 project and application	6
1.2.1 Create new MyLab1 project	6
1.2.2 Create MyLab1 application's main page	11
1.2.3 Create cameraPage for Add Memory	15
1.3 Add iPhone environment and preview in Mobile Browser Simulator	22
1.3.1 Add iPhone environment	23
1.3.2 Preview in Mobile Browser Simulator	24
1.4 Implement a Worklight Adapter	26
1.4.1 Create a Worklight Adapter	26
1.4.2 Test Worklight Adapter	29
1.5 Examine the fully-built and styled version of MyMemories	31
1.5.1 Import and test the completed MyMemories application	32
1.5.2 Locate and Review the Styling of the complete app	32
1.5.3 Locate and Review the Cordova Camera API implementation	33
1.5.4 Locate and Review the JSONStore implementation	34
1.5.5 Locate and Review the adapter invocation	35

1. Getting Started with IBM Worklight

This Proof of Technology asset contains 2 labs. In Lab 1 you will familiarize yourself with building and testing a mobile application using Worklight Studio. In Lab 2 you are going to continue to work with the application in the Worklight Studio, but you will also work directly with the Worklight Server and the Application Center.

In this lab you will develop a basic mobile app using the IBM Worklight Studio development environment. You will use cross platform techniques such as HTML5, CSS3, JavaScript and the jQuery Mobile framework. The app you will develop and use throughout this POT is called **MyMemories** and it provides two features in the mobile app – 1) taking a picture of a scene and a brief note with title to save it to a list of MyMemories items 2) accessing the list of saved MyMemories items to see detail information. Saved items will be automatically tagged with geo-location information. Below are some screenshots from the fully developed application:



In this lab, you will learn how to do the following:

- Create a new Worklight project and a Worklight application in Worklight Studio
- Use Rich Page Editor (RPE) to add UI elements to application
- Build and deploy a Worklight application to test server in Worklight Studio
- Add an environment for iPhone
- Use Preview feature to preview and test application in the Mobile Browser Simulator
- Create a Worklight Adapter (HTTP)
- Invoke Worklight Adapter from Worklight Studio
- Import a fully-customized and functional version of the MyMemories application

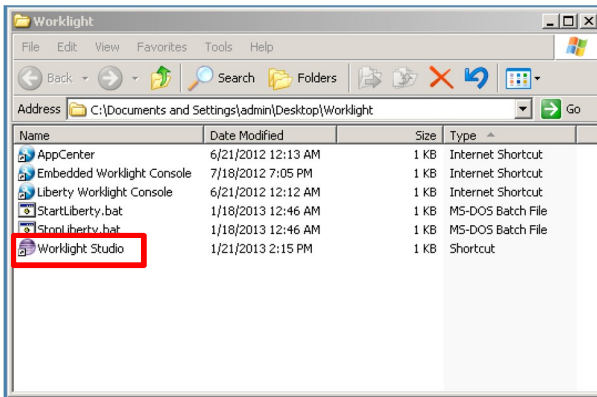
- Review additions – CSS, Cordova Geolocation API, Cordova Camera API, JSONStore
- Preview the fully-functioning application

1.1 Start Worklight Studio

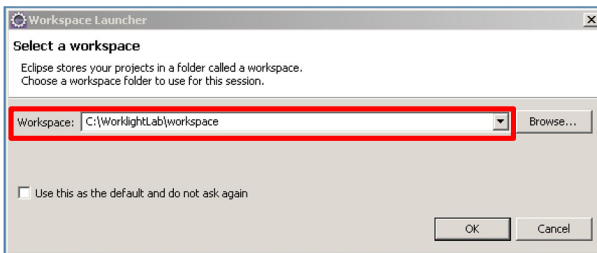
This lab assumes that you have obtained and started the corresponding VMWare image. In the image you will launch Eclipse with the Worklight Studio tooling and then create a new project called MyLab1, with a new app also called MyLab1. This app will be used until you import fully-built and styled version of the MyMemories project and app in section 1.5 in order to avoid any conflicts on project and app names.

1.1.1 Start Worklight Studio

__1. In Worklight folder, double-click the **Worklight Studio** icon.



__2. On the Workspace Launcher dialog accept the default workspace path **C:\WorklightLab\workspace** and click **OK**.

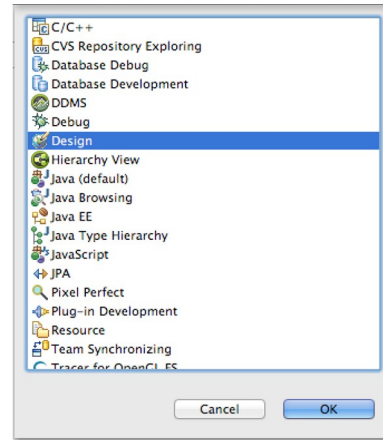
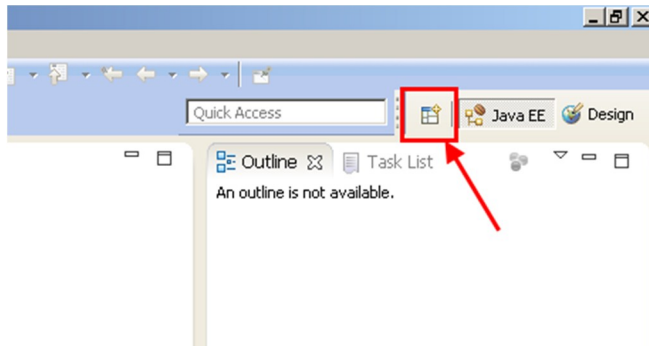


__3. If you receive an Eclipse Welcome Screen, dismiss it by **closing** the *Welcome* tab.

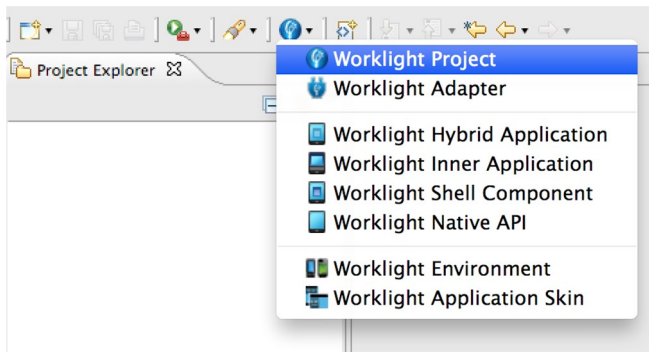
1.2 Create new MyLab1 project and application

1.2.1 Create new MyLab1 project

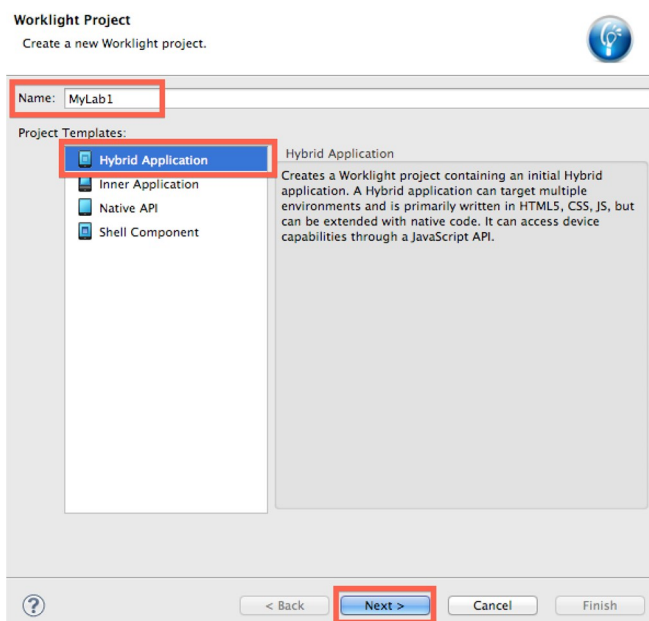
__1. Switch to the Design perspective by selecting the **Design** perspective option if visible, or use the **Open perspectives icon** and selecting **Other > Design**.



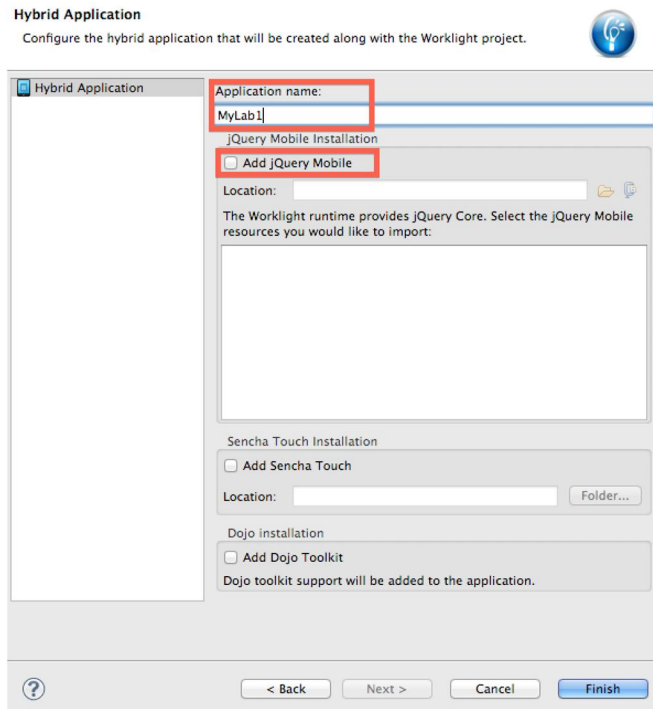
___2. Go to Worklight icon in toolbar on top, and select **Worklight Project**.



___3. In New Worklight Project window, enter MyLab1 in Name field, select Hybrid Application in Project Templates section, and click Next button.



- ___4. Enter MyLab1 in Application name field and click on Add jQuery Mobile checkbox in jQuery Mobile Installation section. Select folder icon right next to Location input field. The JQuery Mobile parts can found in the C:\WorklightLab\imports\lab1\ folder (they may need to be unzipped before importing).



- ___5. Find and select jquery.mobile-1.2.0 folder, and click OK. In the list of folders and files, select images folder, css folder, and js folder, and click Finish.

Hybrid Application

Configure the hybrid application that will be created along with the Worklight project.



Hybrid Application

Application name:
MyLab1

jQuery Mobile Installation

Add jQuery Mobile

Location: /Users/yongkwonrickchoi/Documents/jquery.mc

The Worklight runtime provides jQuery Core. Select the jQuery Mobile resources you would like to import:

- css
 - jquery.mobile-1.2.0.css
- images
- js
 - jquery.mobile-1.2.0.js

Sencha Touch Installation

Add Sencha Touch

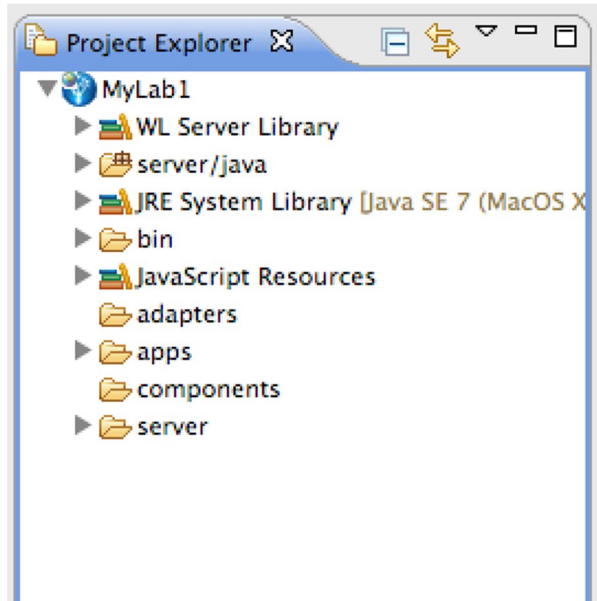
Location: Folder...

Dojo installation

Add Dojo Toolkit

Dojo toolkit support will be added to the application.

- __6. The application template will be populated and the **application-descriptor.xml** file will open by default. Application characteristics such as authentication and server URL are managed in this file. We can leave it at its defaults for now, while we investigate the parts of a Worklight project and application. In the *Project Explorer* pane, expand the **MyLab1** project. Review the folder structure that has been created.



WL Server Library:

Contains the Worklight API jar file

server/java:

Location for server-side java code in java-base adapters (advanced)

JRE System Library:

Contains the JRE used in this project

JavaScript Resources :

Contains the project's JavaScript classes content

adapters:

Contains the project's adapters (used for backend connectivity)

apps:

Contains the project's applications

bin:

Location for build artifacts (wlap files) that are deployed to a Worklight server

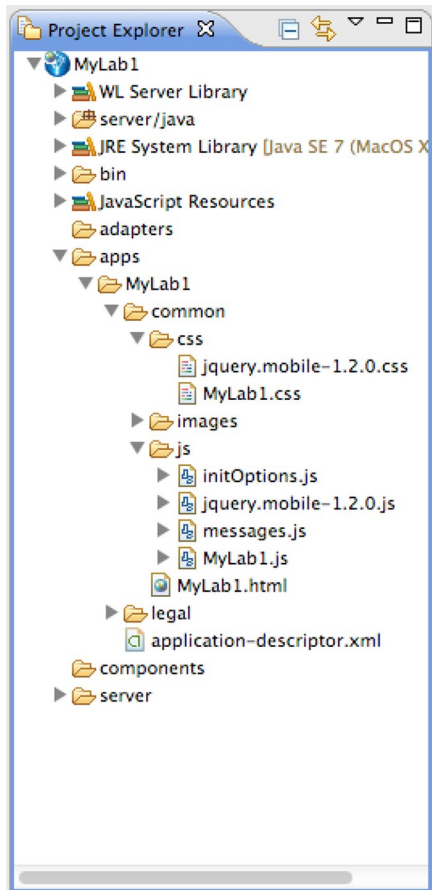
components:

Contains shell application components (advanced)

server:

Contains configuration files and extension locations for the embedded Worklight test server

- ___7. In the *Project Explorer*, expand the **apps** folder then the **apps > MyLab1** folder and the apps > MyLab1 > common folder that were created by the new application wizard.



common: the default 'environment' that gets created for an application.

css: **MyLab1.css** – the main application CSS file,

images: Default images for the common environment.

js: **MyLab1.js** – the main JavaScript file for the app,
messages.js – JSON object holding all app messages,
initOptions.js – initialization options for the app.

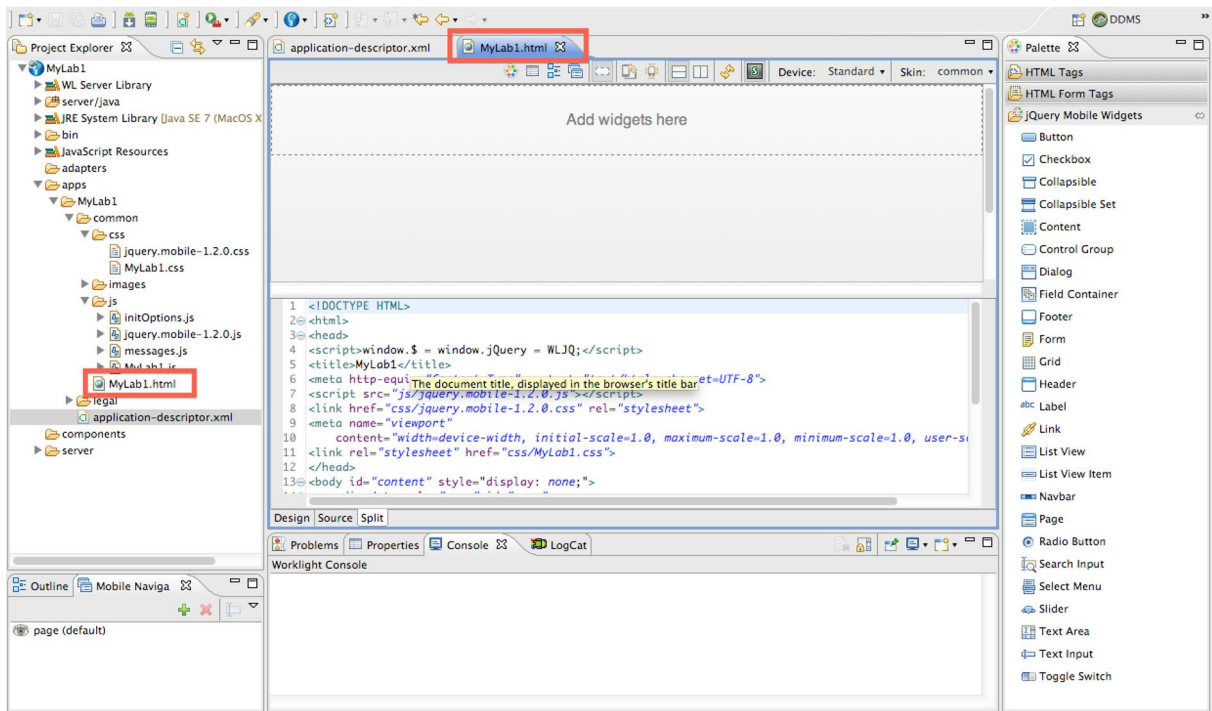
MyLab1.html: The main application html file.
 Application can have multiple html files.

legal: All legal related documents.

application-descriptor.xml: Application's meta data
 (security config, server url, etc.)

1.2.2 Create MyLab1 application's main page

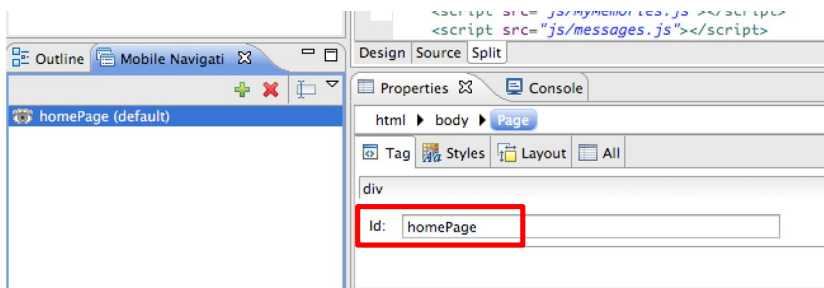
- ___1. In Project Explorer, double-click MyLab1.html to open and edit the automatically generated main page.



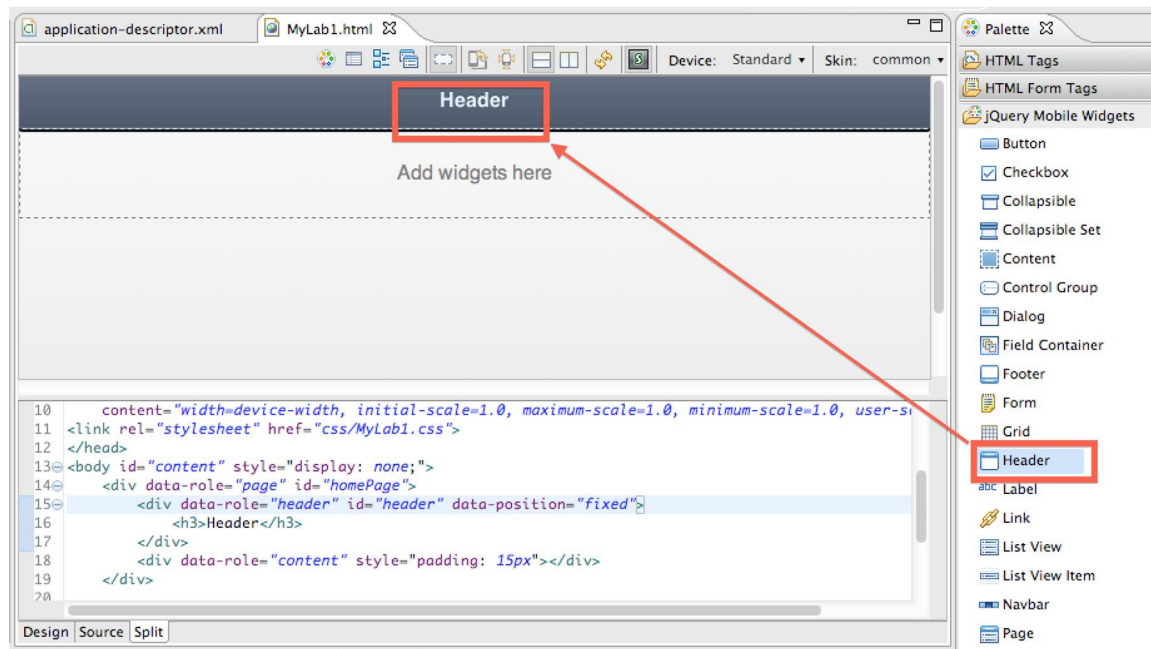
__2. If the Mobile Navigation window is not open in your eclipse workspace, you should open it using the Show/Hide Mobile Navigation icon on the Rich Page Editor toolbar



__3. Select page (default) in Mobile Navigation window, go to Properties and select Page → Tag. In Id input field, change the value page to homepage.



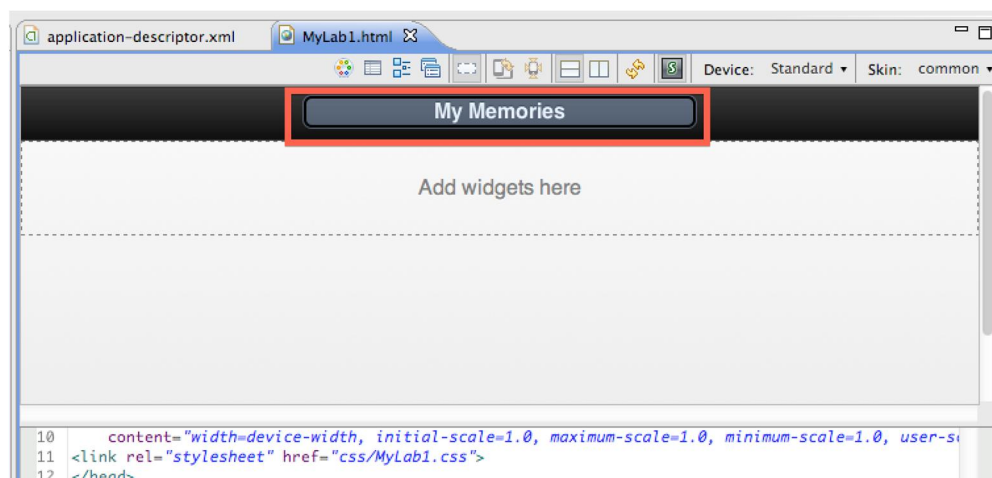
__4. In the Palette of jQuery Mobile Widgets, select & drag Header widget and drop it into main editor window on the top section (above “Content” – the editor will prompt for correct location).



TIP: If at any time you find the design view editor to be not accepting your locations as well as you'd like, you can refresh the design view using the Refresh page icon and it should be more predictable.

NOTE: You can also drop directly into the correct source location in the source view if that gives you a better sense of control for where the code is being inserted.

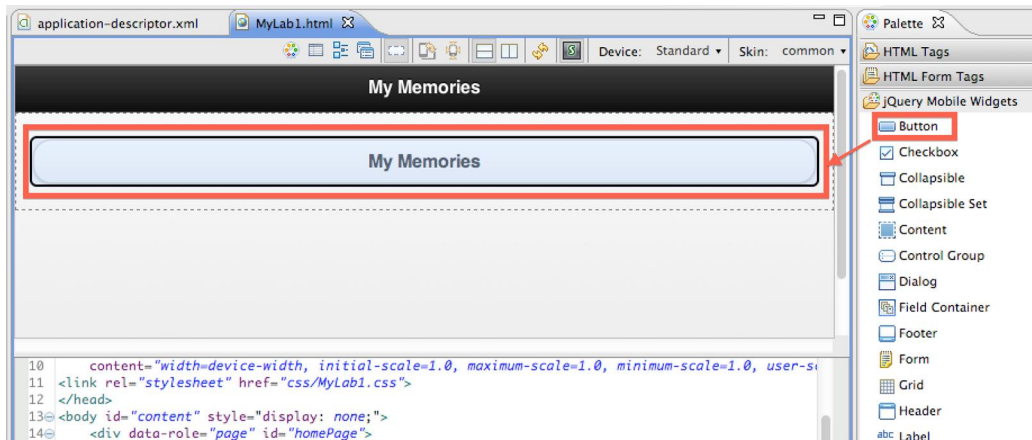
- __5. Change the word Header to My Memories by double-clicking it in the header widget.



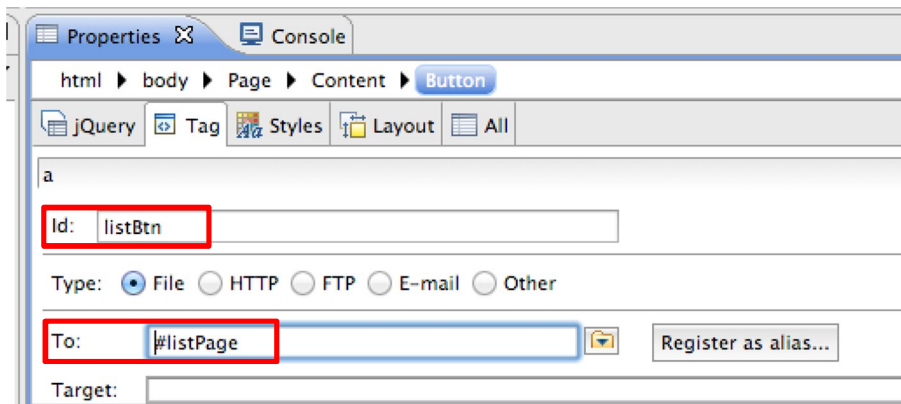
- __6. In the jQuery Mobile Widgets section of the Palette, select & drag the 'Button' widget and drop it into main editor window, right under Header section.

NOTE: Make sure you are selecting from the the JQuery Mobile Widgets section of the Palette. There are multiple 'Button' widgets in the different categories of widgets which will not provide the expected JQuery syntax and functionality.

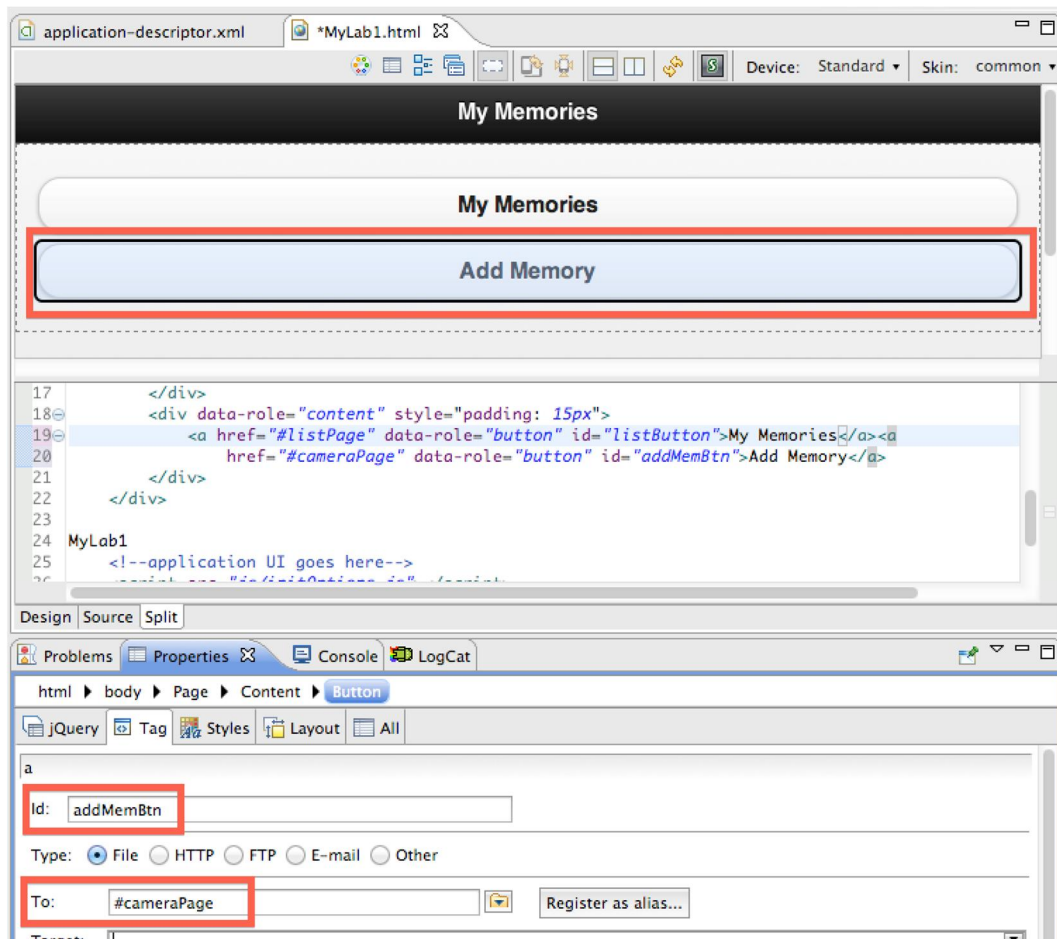
- __7. Change the word 'Button' to 'My Memories', by double-clicking it in the button widget and typing the new name. (you can also make this change in the source view).



- __8. In Properties windows, change Id value button to listBtn and also change To value to #listPage. In jQuery tab, set Theme to d.

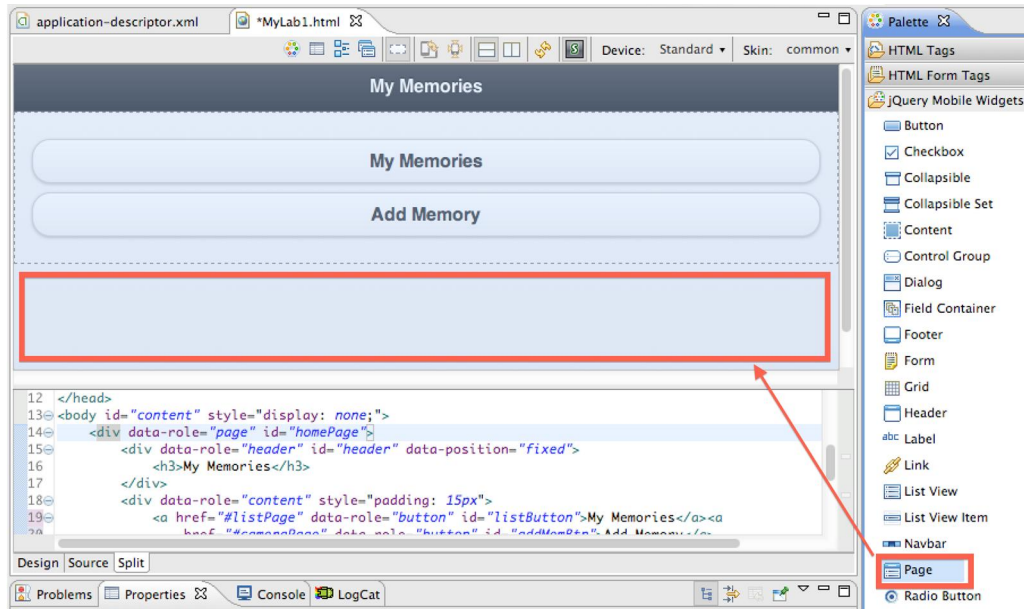


- __9. In the same way that you did for My Memories button, add an additional button widget right under My Memories button. Drag & drop Button widget under My Memories button, change the word Button to Add Memory, change Id value button to addMemBtn, change To value to #cameraPage and set Theme to d in jQuery tab.

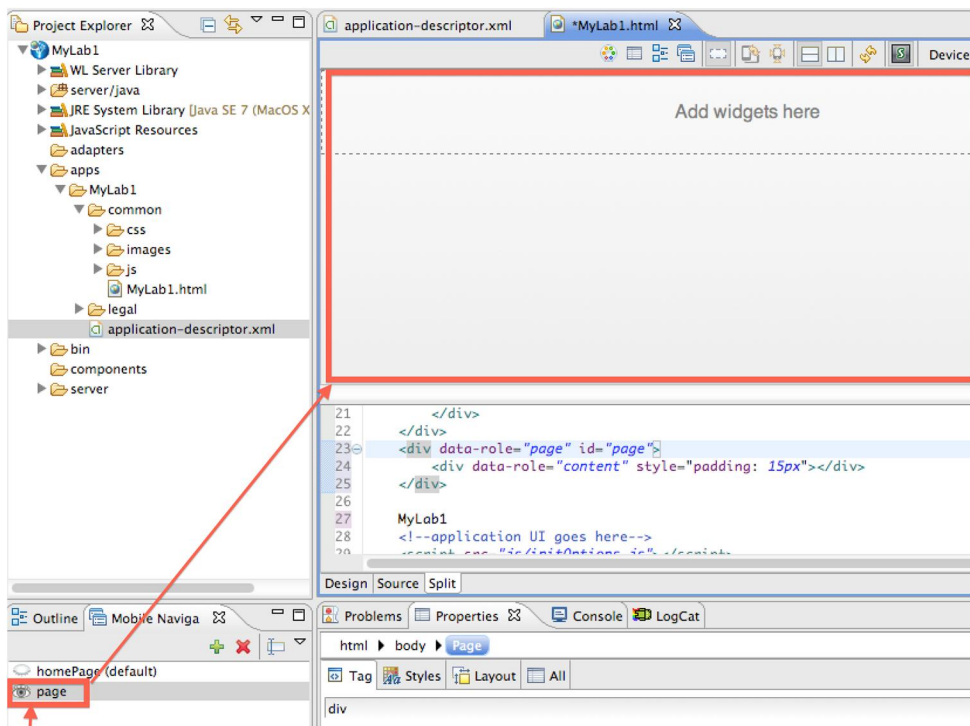


1.2.3 Create cameraPage for Add Memory

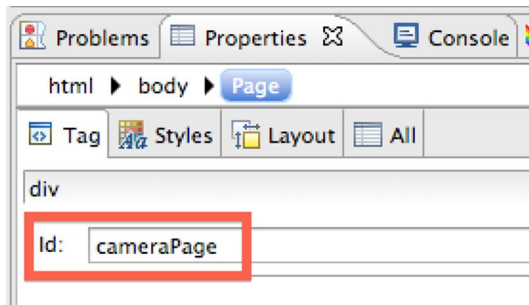
- ___1. In the Palette of jQuery Mobile Widgets, select & drag Page widget and drop it into main editor window right under the widgets of homePage. (Insert below page 'homePage').



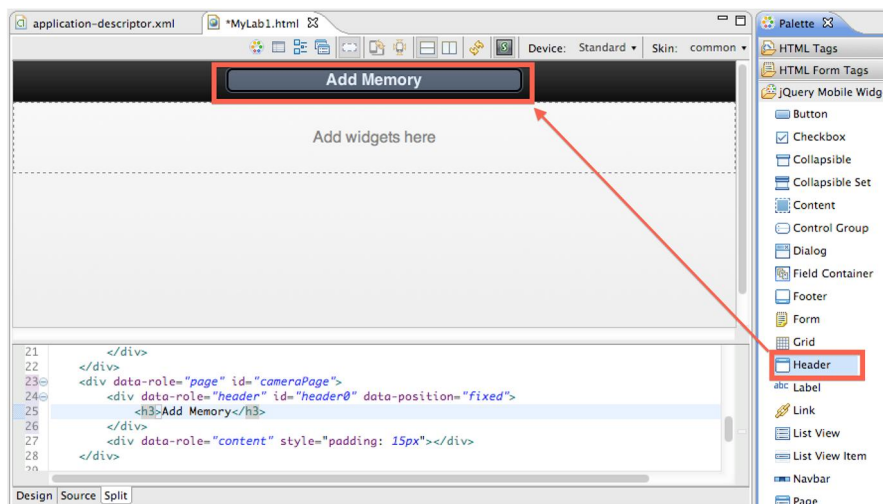
2. In Mobile Navigation window, select the small icon for the new page named 'page' to change the design view to the newly inserted page.



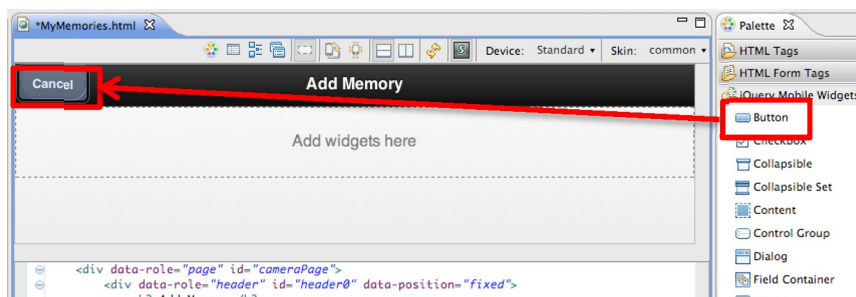
3. Go to Properties and select Page → Tag. In Id input field, change the value page to cameraPage, just like what you did on homePage.



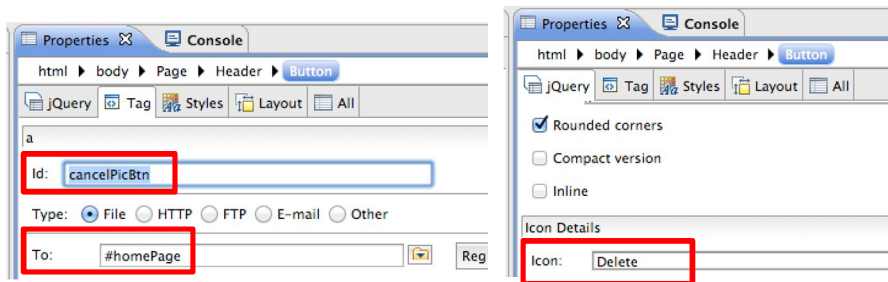
- __4. In the mobile navigation view, double click on the newly created page to ensure you are editing the right page. In the Palette of jQuery Mobile Widgets, select & drag Header widget and drop it into main editor window on the top section, above 'Content' just as on homePage, and change the word Header to Add Memory by double-clicking it in the header widget.



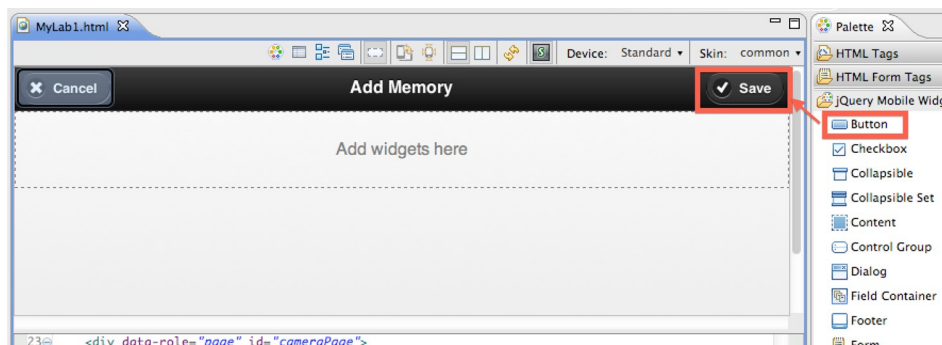
- __5. In the Palette of jQuery Mobile Widgets, select & drag Button widget and drop it into Header section (before heading), and change the word 'Button' to 'Cancel'.



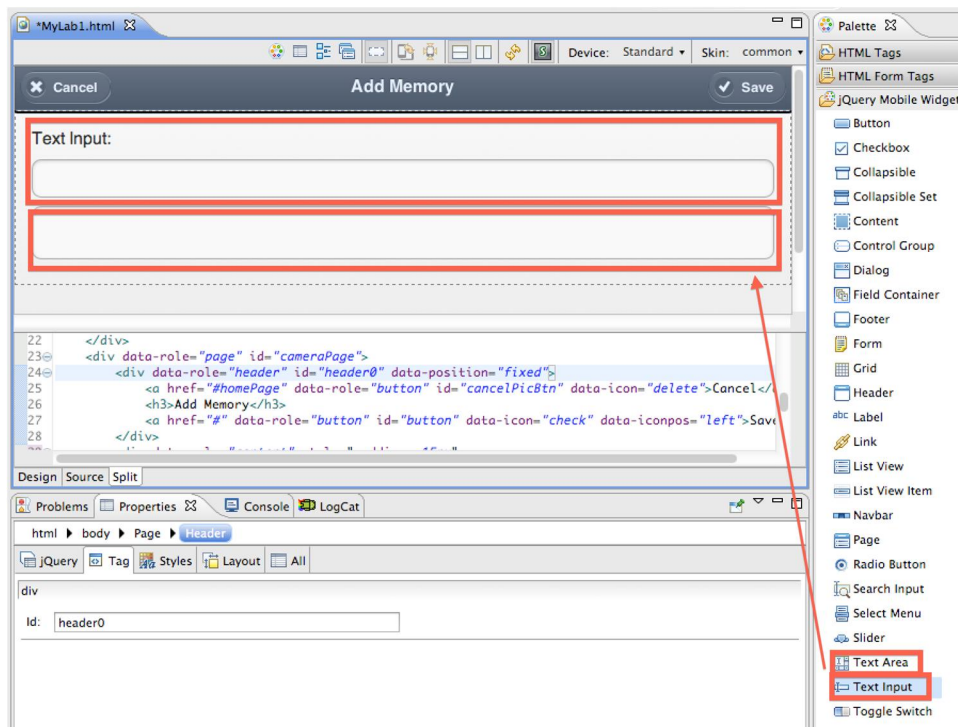
- __6. In the Properties window, change Id value button to cancelPicBtn and change To value to #homePage in Tag tab. In jQuery tab, set Icon value to Delete to add an "X" icon next to the Cancel label.



7. In the Palette of jQuery Mobile Widgets, select & drag Button widget and drop it into Header section again (after heading), and change the word Button to Save, just like you did on Cancel button. In the Properties window, change Id value button to savePicBtn in Tag tab. In jQuery tab, set Icon value to Check to add a checkmark icon to the Save label.



8. In the Palette of jQuery Mobile Widgets, select & drag the Text Input widget and drop it into Content section (below header) and then do the same with Text Area widget (right below Text Input).



NOTE: Depending on the screen resolution of the workstation you are using, and the orientation of the design/source editors (horizontal, vertical or individual), you may see different visual treatments given the amount of space available to position the items. Use the different view options to get the most comfortable editing environment for your own style.

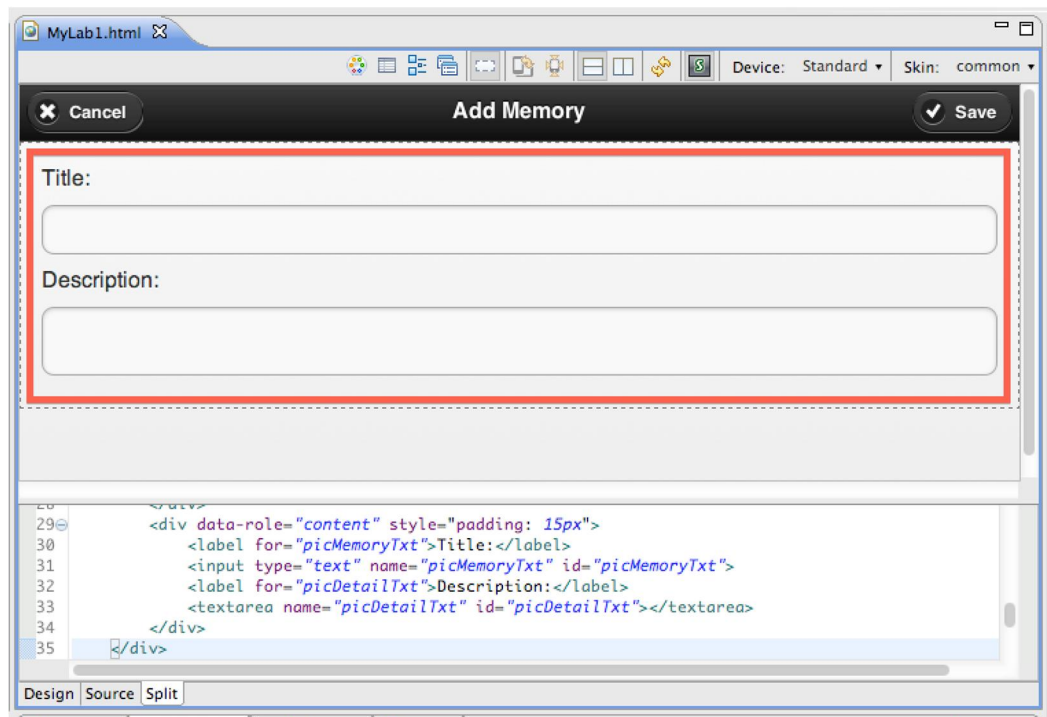
__9. In Source window, replace the following HTML code:

```
<label for="text">Text Input:</label>
<input type="text" name="text" id="text">
<textarea id="textarea"></textarea>
```

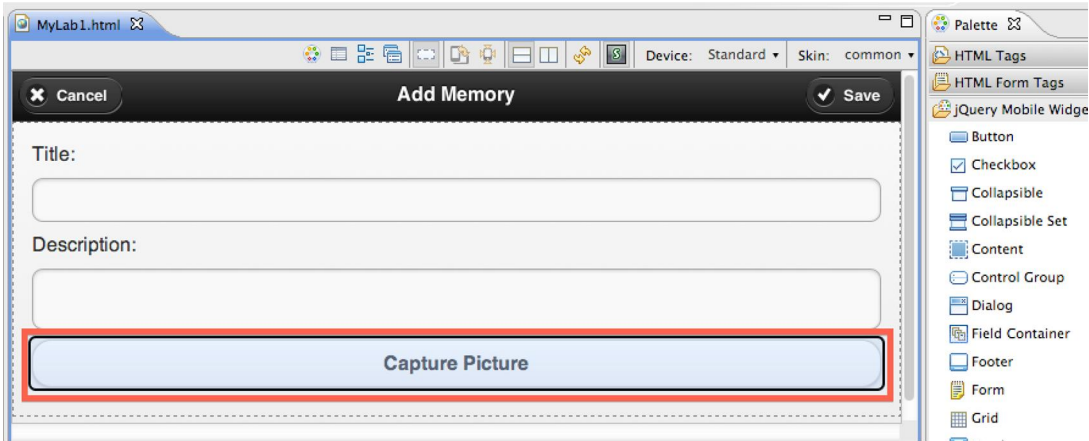
to this (available for copy & paste from C:\WorklighLabs\imports\lab1\Lab1_snippets.txt):

```
<label for="picMemoryTxt">Title:</label>
<input type="text" name="picMemoryTxt" id="picMemoryTxt">
<label for="picDetailTxt">Description:</label>
<textarea name="picDetailTxt" id="picDetailTxt"></textarea>
```

The widgets should change to the following:



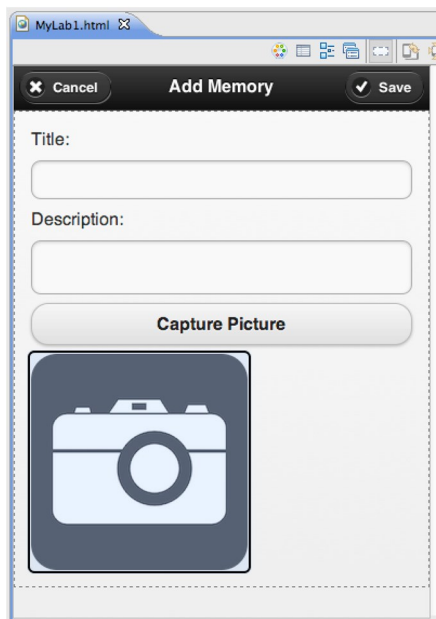
- ___10. In the Palette of jQuery Mobile Widgets, select & drag the Button widget and drop it into Content, below the Text Area, and change the word Button to Capture Picture, just like what you did on Cancel and Save button widgets.



- ___11. Copy/import the camera_icon.png file from C:/WorklightLab/imports/lab1/camera_icon.png into the MyLab1/apps/MyLab1/common/images folder
- ___12. In Source window, insert the following HTML code right under Capture Picture button widget code:

```
<div class="capturedImage" id="capturedImage">
<img id='captareImageImg' src='images/camera_icon.png'
height='200px' width='200px' style='display:block'> </div>
```

And it shows the following:



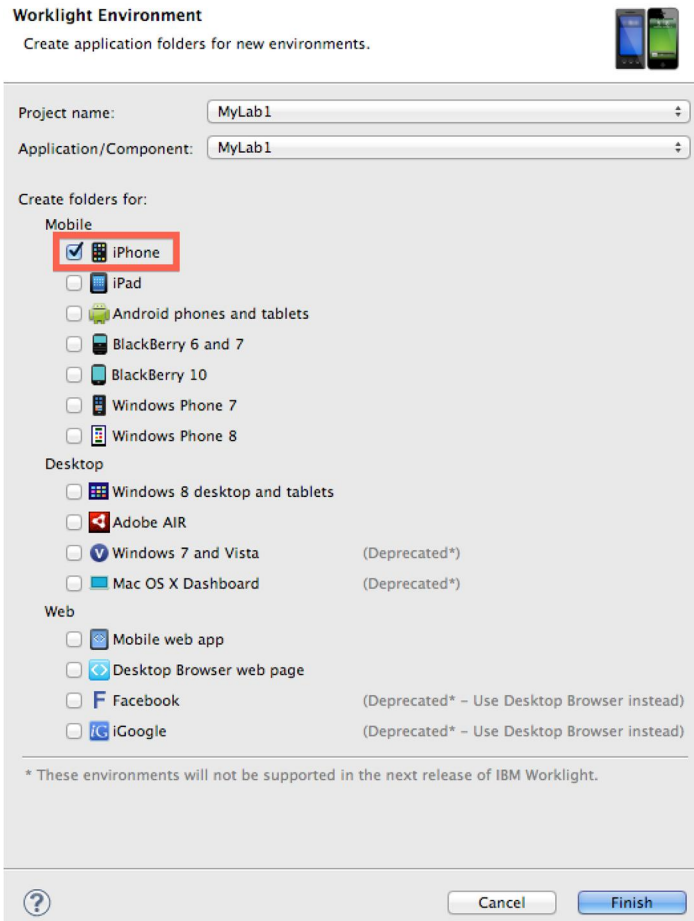
__13. Save changes MyLab1.html (Ctrl-s) and any other open source files.

1.3 Add iPhone environment and preview in Mobile Browser Simulator

Worklight's cross-platform support is based on the concept of environments. In this section we will enable our app to be built for the iPhone environment, and test the iPhone version in the Mobile Browser Simulator.

1.3.1 Add iPhone environment

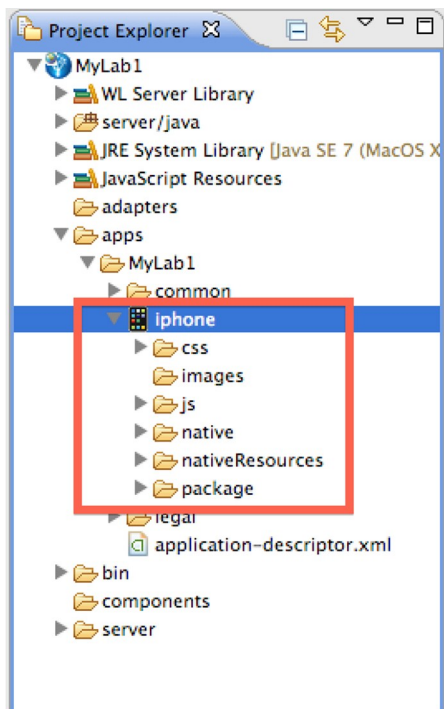
- ___1. Highlight any element of the MyLab1 project in the Project Navigator, and select New Worklight Environment (from the 'New' element in the right click menu or from 'File' on the menu bar).
- ___2. Select the iPhone environment (checkbox)



- ___3. Click 'Finish' to dismiss the selection panel and generate the iPhone environment folder

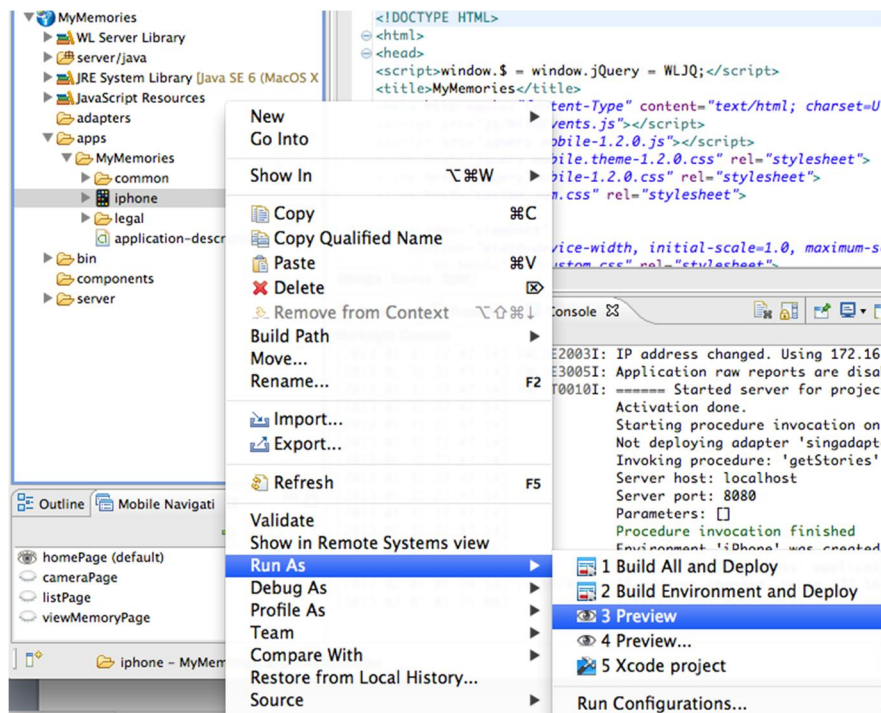
Note: If you check the Console tab, you should see messages about the iPhone Environment being created, the build process being started, and a message that the build finished for the iPhone environment.

- ___4. Examine the new iPhone folder (MyLab1->apps->MyLab1->iPhone)
 - This is where iPhone specific customizations and native content can be added to the project.



1.3.2 Preview in Mobile Browser Simulator

- __1. Select the iPhone environment folder
- __2. Right click and select Run As -> Preview (first preview option in the list)

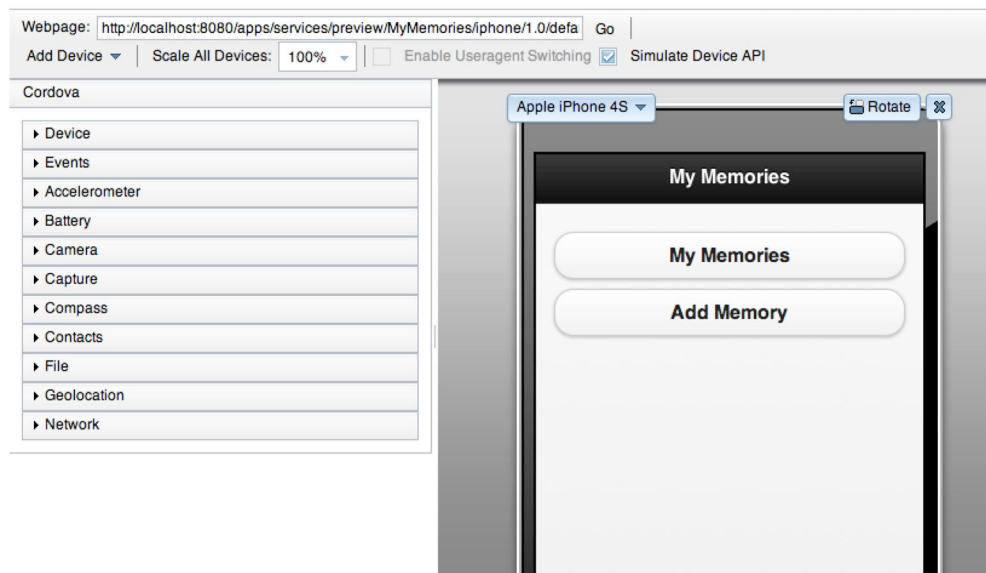


- ___3. Preview will launch the app in the Mobile Browser Simulator, with an iPhone skin and Cordova controls.

NOTE: The VMware image should be pre-configured to launch the Simulator in an external browser (Chrome, Safari or Firefox). If it opens within eclipse, you can go to Eclipse Preferences, General->Web Browser, and change to one of the external options. Using MBS in an external browser provides more screen space and a better experience for using the Cordova controls and Web Inspector/Firebug.

Mobile Browser Simulator

The Mobile Browser Simulator displays mobile web pages in a variety of mobile browser sizes and shapes.



NOTE: The Mobile Browser Simulator (MBS) is an excellent resource for debugging Worklight Hybrid applications directly on the developer's desktop. With full access to browser features like Web Inspector/Firebug, you can view the resources in your application, set breakpoints, manipulate CSS – all of the programming debug/diagnostic tasks that are common to web applications are available in MBS.

- ___4. Navigate the application screens within the Mobile Browser Simulator. Notice things like
- The MBS selector for iPhone type,
 - The various Cordova device controls that can be set to values and you can see represented in your application (when you implement them – we will see this shortly).
 - If you are familiar with Web Inspector/Firebug, attempt to Inspect Element on the MBS screen and start looking at how you are able to do common HTML, JS, CSS debugging and manipulation within your mobile app.

1.4 Implement a Worklight Adapter

In this section we will create a Worklight Adapter to invoke a cloud-based geocoding service (google, etc...) and convert lat/long coordinates from the device's location service (GPS or triangulation) into a formatted address for use with our memory. This will show the simplicity and flexibility of the Worklight adapter model.

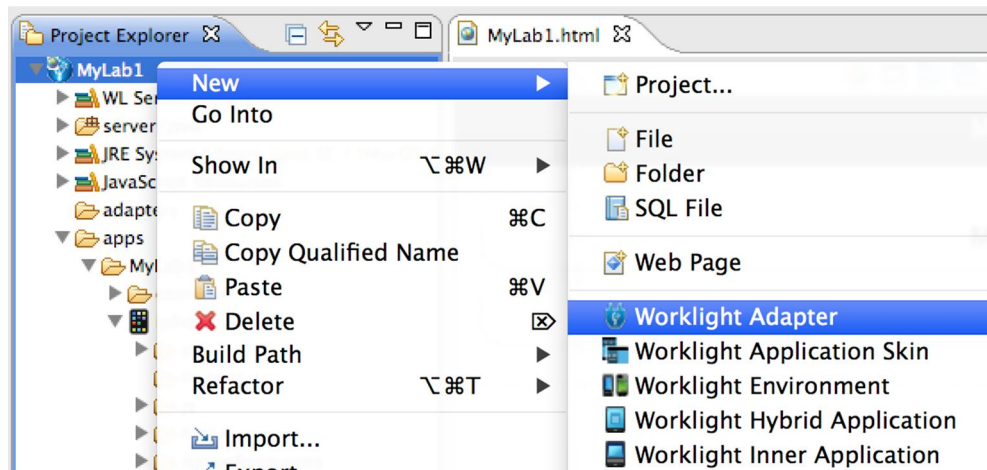
The Worklight Adapter framework makes use of JSON to optimize the data being passed back to the client and simplifies the client-side programming model. It would be quite possible to allow the device to retrieve coordinates directly from the cloud service, but using the Worklight Adapter model allows for immediately switching service providers without impacting apps on devices, as well as full integration with the Worklight security model when applicable. A Worklight Adapter consists of an Adapter XML file – the descriptor for the type of service being contacted (HTTP Web/REST services, SQL, Castiron, or JMS), and an adapter implementation file – a Javascript module that assembles the URL path (from parameters passed in) and any filtering or manipulation of the data returned from the backend service.

In a subsequent section, we will demonstrate how the client invokes the adapter, and how to use the returned data within the application.

1.4.1 Create a Worklight Adapter


In this section, we will create a Worklight Adapter called GPSLocator to translate location information from latitude and longitude into a real address using Google's geocoding service over HTTP.

1. In Project Explorer window, select and second-mouse-click on MyLab1 project. Select New → Worklight Adapter.



- ___2. In the form, select HTTP Adapter for Adapter type, type GPSLocator in Adapter name input field, and click Finish button.

Worklight Adapter
Create a new adapter.



Project name:

Adapter type:

Adapter name:

Create procedures for offline JSON store

Retrieve JSON data with:

Add JSON data with:

Replace JSON data with:

Remove JSON data with:

- ___3. Once you finish step 2 above, you will see GPSLocator.xml Design view window in Worklight Studio, and remove Procedure “GetStoriesFiltered” from Overview section.

Adapter Editor

Overview

type filter text

- Adapter "GPSLocator"
 - Connectivity
 - Connection Policy
 - Procedure "getStories"
 - Procedure "getStoriesFiltered"

Add...
Remove
Up
Down

Details

Name*:
The name of the procedure. This name must be unique name within the adapter. It can contain alphanumeric characters and underscores, and must start with a letter

Display name:

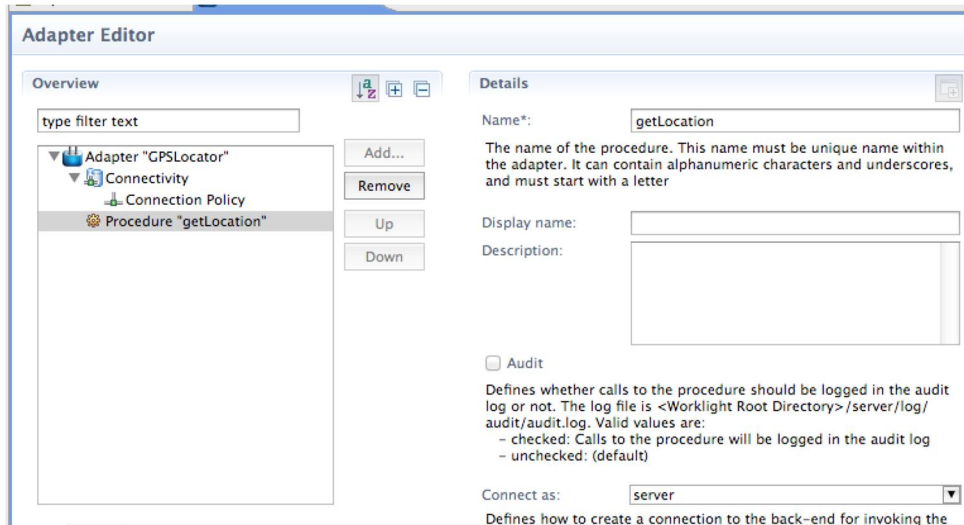
Description:

Audit
Defines whether calls to the procedure should be logged in the audit log or not. The log file is <Worklight Root Directory>/server/log/audit/audit.log. Valid values are:
- checked: Calls to the procedure will be logged in the audit log
- unchecked: (default)

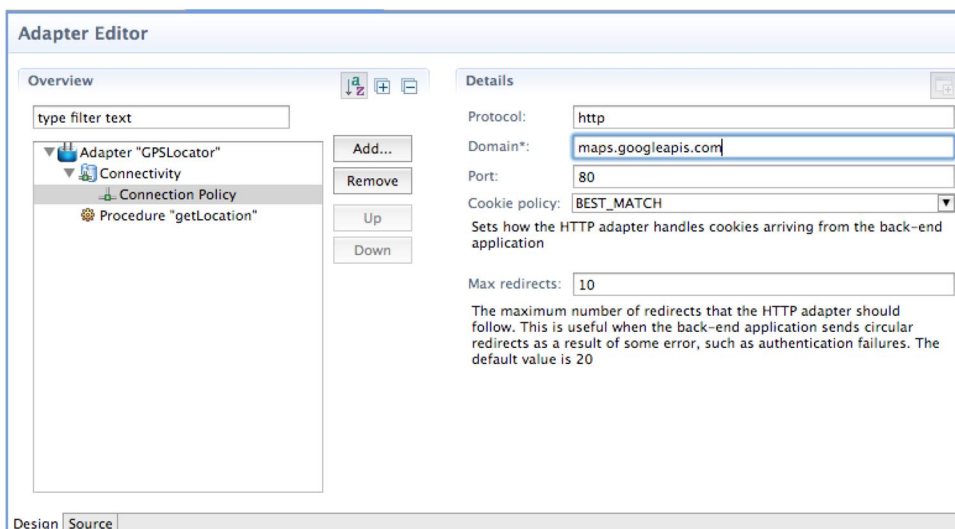
Connect as:
Defines how to create a connection to the back-end for invoking the

Design | Source

- ___4. Select Procedure “getStories”, and change the name to getLocation in Details section.



- __5. Go to Adapter “GPSLocator” → Connectivity → Connection Policy, and change Domain value from rss.cnn.com to maps.googleapis.com. Save GPSLocator.xml file.



- __6. In the Package Explorer window, open GPSLocator-impl.js file and replace all the content in the file with the following codes and save the file (available for copy & paste from C:\WorklightLab\imports\lab1\Lab1_snippets.txt):

```
// Google Map API call

function getLocation(gpsLat, gpsLong) {

path = "/maps/api/geocode/json?latlng=" + gpsLat + "," + gpsLong +
"&sensor=false";

var input = {

    method : 'get',
```

```

    returnedContentType : 'json',

    path : path
};

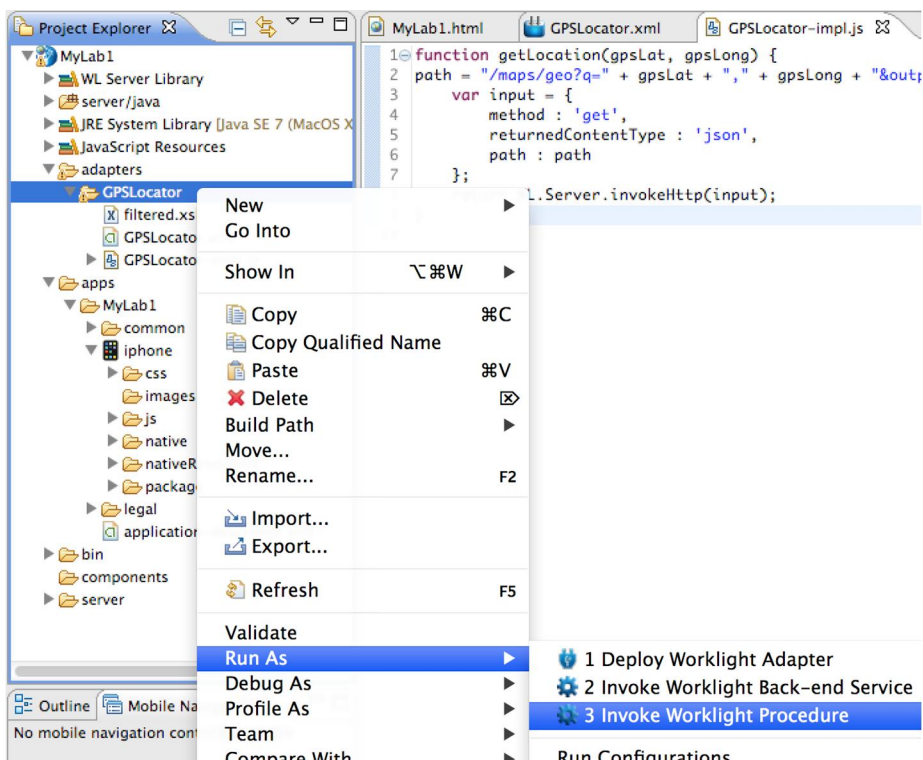
return WL.Server.invokeHttp(input);
}

```

1.4.2 Test Worklight Adapter

In this section, we will learn how to invoke the adapter in Worklight Studio to see if the adapter works properly, and how to use the returned data within the application.

1. In Project Explorer window, select and second-mouse-click on GPSLocator folder, Run As → Invoke Worklight Procedure.



2. In Edit configuration and launch window, select getLocation for Procedure name and enter "30.377047,-91.123484" (without quotation marks) into Parameters section, and click Run button.

Edit configuration and launch.



Name: Invoke Procedure MyLab1 - GPSLocator

Invoke Procedure Data

Project name: MyLab1

Adapter name: GPSLocator

Procedure name: getLocation

Signature:
getLocation (gpsLat, gpsLong)

Parameters (comma-separated):
30.377047,-91.123484

Apply Revert

Close Run

- In Invoke Procedure Result windows, check if the returned result (from Google Maps) shows correct information with statusCode of 200 and statusReason of "OK".

```

Invocation Result of procedure: 'getLocation' from the Worklight Server:
{
  "Placemark": [
    {
      "AddressDetails": {
        "Accuracy": 8,
        "Country": {
          "AdministrativeAreaName": "LA",
          "SubAdministrativeArea": {
            "Locality": {
              "DependentLocalityName": "Highlands\Perkins",
              "PostalCode": {
                "PostalCodeNumber": "70808"
              },
              "Thoroughfare": {
                "ThoroughfareName": "524 Magnolia Wood Avenue"
              }
            },
            "LocalityName": "Baton Rouge"
          }
        }
      }
    }
  ]
}
    
```

```

MyLab1/apps/MyLab1/common/MyLab1.html
Invocation Result of procedure: 'getLocation' from the Worklight Server:
{
  "Status": {
    "code": 200,
    "request": "geocode"
  },
  "isSuccessful": true,
  "name": "30.377047,-91.123484",
  "responseHeaders": {
    "Access-Control-Allow-Origin": "*",
    "Cache-Control": "private",
    "Content-Type": "text/javascript; charset=UTF-8",
    "Date": "Tue, 05 Mar 2013 02:49:56 GMT",
    "Server": "mfe",
    "Transfer-Encoding": "chunked",
    "Vary": "Accept-Language",
    "X-Frame-Options": "SAMEORIGIN",
    "X-XSS-Protection": "1; mode=block"
  },
  "statusCode": 200,
  "statusReason": "OK"
}
    
```

- Close all open editor windows.

1.5 Examine the fully-built and styled version of MyMemories

In this portion of the lab we will import a pre-built version of MyMemories, complete with themes and styles. We will explore some of the advanced Worklight features, such as Cordova API calls (for camera, gps) and the JSONStore – a JSON-based encrypted storage feature of Worklight that can store JSON data locally or utilize adapters to provide synchronization with a back-end data source.

1.5.1 Import and test the completed MyMemories application

In this step, we will import a copy of the MyMemories application which has been richly stylized to present a vibrant and professional look and feel. The styles are added through a combination of CSS and JQuery Mobile options.

- __1. From Eclipse, Select File→Import from the menu bar
- __2. Select **Existing Projects into Workspace** and hit **Next**
- __3. Use the 'Select archive file' option and **Browse** to the project zip in
C:\WorklightLab\imports\lab1\MyMemories.zip
- __4. Hit **Finish** to import the project
- __5. Once the project appears in Eclipse, expand to MyMemories/apps/MyMemories, right click and select **Build and deploy all**
- __6. Preview in Mobile Browser Simulator

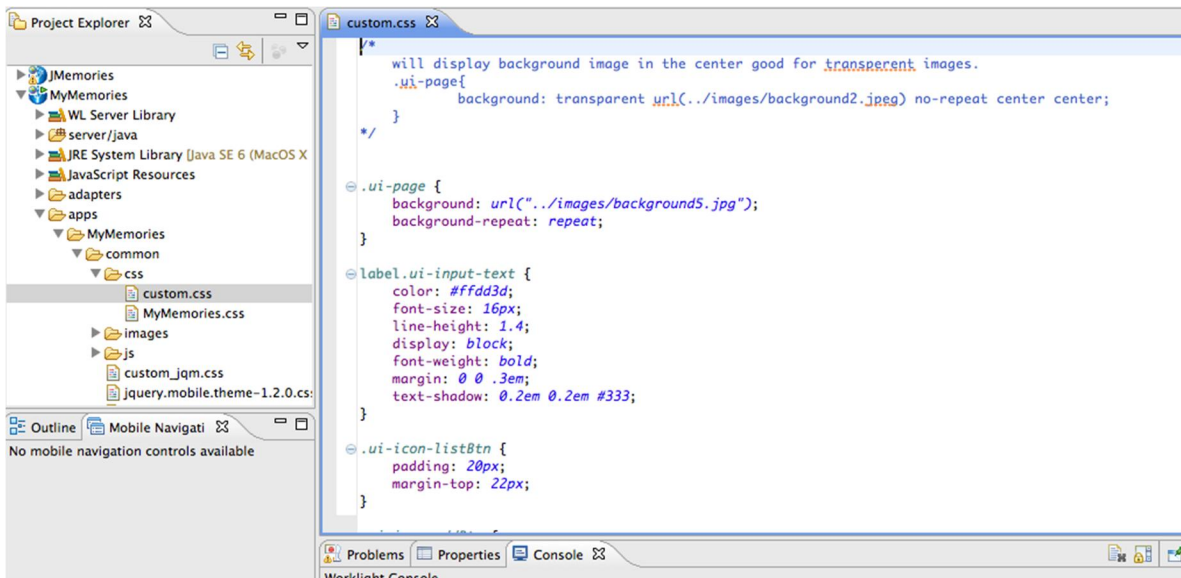
In the next steps we will explore the app, noting the key changes involved:

- Styling – Themed background, buttons, text colors have been added
- Camera – the code will request a photo from the Cordova camera API
- JSONStore implementation – code has been added to use the JSONStore for client-side storage of the memories list
- Geolocation – The worklight adapter has been enabled with invocation logic in the client application to obtain the current location coordinates from the Cordova location service (GPS) and leverage the adapter to translate those coordinates into address details (internet required)

1.5.2 Locate and Review the Styling of the complete app

Most of the visual changes were implemented with custom CSS.

- __1. In the Worklight Studio, navigate to MyMemories/apps/MyMemories/common/css in the Project Explorer view and open the custom.css and MyMemories.css files.



CSS (Cascading Style Sheets) is the primary way you affect the design of HTML. With CSS you can apply colors, shading, images, fonts, spacing and other visual changes. If you have web styling experience you will recognize the CSS tags that are adding background images, colors and other styling features to the newly imported version of the MyMemories app. Explore as necessary and feel free to ask the instructors to explain anything you don't understand.

- __2. Advanced: You can also return to the Mobile Browser Simulator and use web inspector to review the html, js and css behaviors. Right click any blank space in the browser and select Inspect Element from the pop-up menu.

1.5.3 Locate and Review the Cordova Camera API implementation

The new version of the app has implemented the Cordova camera api to request that the device open the camera, allow the user to take a snapshot, and return that snapshot to the application.

- __3. The camera api calls are contained within the MyMemories/apps/MyMemories/js/MyMemories.js file. Navigate to that file and open it in Worklight Studio. Locate the block below to see the entire set of code required to handle photos:

```

// Capture the image
function capturePhoto() {
  console.log("in capturePhoto()");
  // Take picture using device camera and retrieve image as base64-encoded string
  navigator.camera.getPicture(onPhotoDataSuccess, onFail, { quality: 10,
    destinationType: navigator.camera.DestinationType.DATA_URL });
}

//success callBack.
function onPhotoDataSuccess(imageData) {
  console.log("in onPhotoDataSuccess()");
  var smallImage = $('#captureImageImg');
  $('#captureImageImg').attr('src', "data:image/jpeg;base64," + imageData);
  smallImage.src = "data:image/jpeg;base64," + imageData;
  smallImage.show();
}

//failed callBack.
function onFail(message) {
  console.log("in onFail() - " + message);
  alert('Failed because: ' + message);
}

```

- The camera is invoked with the `navigator.camera.getPicture()` function.
- `getPicture()` provides success and failure callback methods for the api layer to return appropriate response data.
- `onPhotoDataSuccess()` handles a successful image capture by assembling the image properties and displaying it.
- `onFail()` handles a failure from the camera by posting a message to the log and an alert to the screen.

1.5.4 Locate and Review the JSONStore implementation

The JSONStore is an encrypted on-device data storage facility provided by the Worklight runtime client. It can be used to easily store and retrieve large volumes of JSON data, such as that returned by a Worklight adapter. Local changes can also be synchronized back to the original datasource via a Worklight adapter. In our lab we are using it for local storage of our memory list content.

- ___1. Locate the file `MyMemories/apps/MyMemories/common/js/memoryStore.js` and open it in the studio. There are a number of functions defined to manipulate the data collection – initialization, listing contents, adding to, removing from, etc...
- ___2. The functions in `memoryStore.js` are invoked by logic in the `MyMemories.js` file in the same directory. If you are interested in JSONStore operations, open `MyMemories.js` and review the interactions between the buttons and functions there, with the JSONStore functions in `memoryStore.js`. A snapshot of several key relationships is below:

```
function wlCommonInit(){
    // Register the click events.
    $('#addMemBtn').bind('click', getGlobalAddress);
    $('#takePicBtn').bind('click', capturePhoto);
    $('#savePicBtn').bind('click', addMemoryToList);
    $(".memItem").live("click", function(){
        loadMemoryToViewPage($(this).attr("id"));
    });
    console.log('finish bind/live elements');
    //initialize the JSON Store collection (Open the store) and load all the memories to the listView.
    initCollection();
}
```

You may notice that we are also using a local variable to shadow the JSONStore “database”. This allows the application to function in the Mobile Browser Simulator, as the JSONStore only runs on real devices.

```
//add memory to the local array (used by the MBS);
localMemoriesList.push(obj);

//add also the memory to the JSONStore.
addMemoryToCollection(obj);
```

Learn more about JSONStore in the Worklight Infocenter and the ‘Getting Started with IBM Worklight’ training modules at <http://ibm.com/worklight>.

1.5.5 Locate and Review the adapter invocation

Previously in this lab, you built an adapter implementation that could be used to fetch data from a back-end datasource. The version of MyMemories that you have imported includes the client-side invocation of that adapter.

- ___1. Locate the function callGeoAdapter() in MyMemories.js (MyMemories/apps/MyMemories/common/js/MyMemories.js)

```

// Called when the adapter completes successfully
var win = function(result){
    globalAddress = result.invocationResult.Placemark[0].address;
    // alert('globalAddress[' + globalAddress.length + '] '+ globalAddress);
    if(globalAddress.length == 0){
        globalAddress = '(Could not translate address via adapter)';
    }
    console.log('-callGeoAdapter:win - address: ' + globalAddress);
    return;
};

// Called when the adapter does NOT complete successfully
var lose = function(){
    console.log('-callGeoAdapter:lose - Failed to obtain address for GPS');
    globalAddress = '(Could not translate address for given GPS)';
    // alert(globalAddress);
    return;
};

var invocationData = {
    adapter    : 'GPSLocator',
    procedure  : 'getLocation',
    parameters : [globalLat, globalLong]
};

var options = {
    onSuccess : win,
    onFailure : lose
};

WL.Client.invokeProcedure(invocationData, options);

```

This block of code contains a success callback method, a failure callback method, sets up invocationData with the adapter, procedure and variables to pass to the backend service and asks the Worklight client to invoke the procedure. These are the necessary components of an client-side adapter invocation. As a related exercise, find the

Lab 1 Complete!

This document was created with Win2PDF available at <http://www.win2pdf.com>.
The unregistered version of Win2PDF is for evaluation or non-commercial use only.
This page will not be added after purchasing Win2PDF.