Fig Leaf Training

Implementing Mobile Web Services with Adobe ColdFusion 10

Session Requirements

Student Files:

http://webapps.figleaf.com/cfjqm.zip

- PDF Workbook
- Lab Exercises

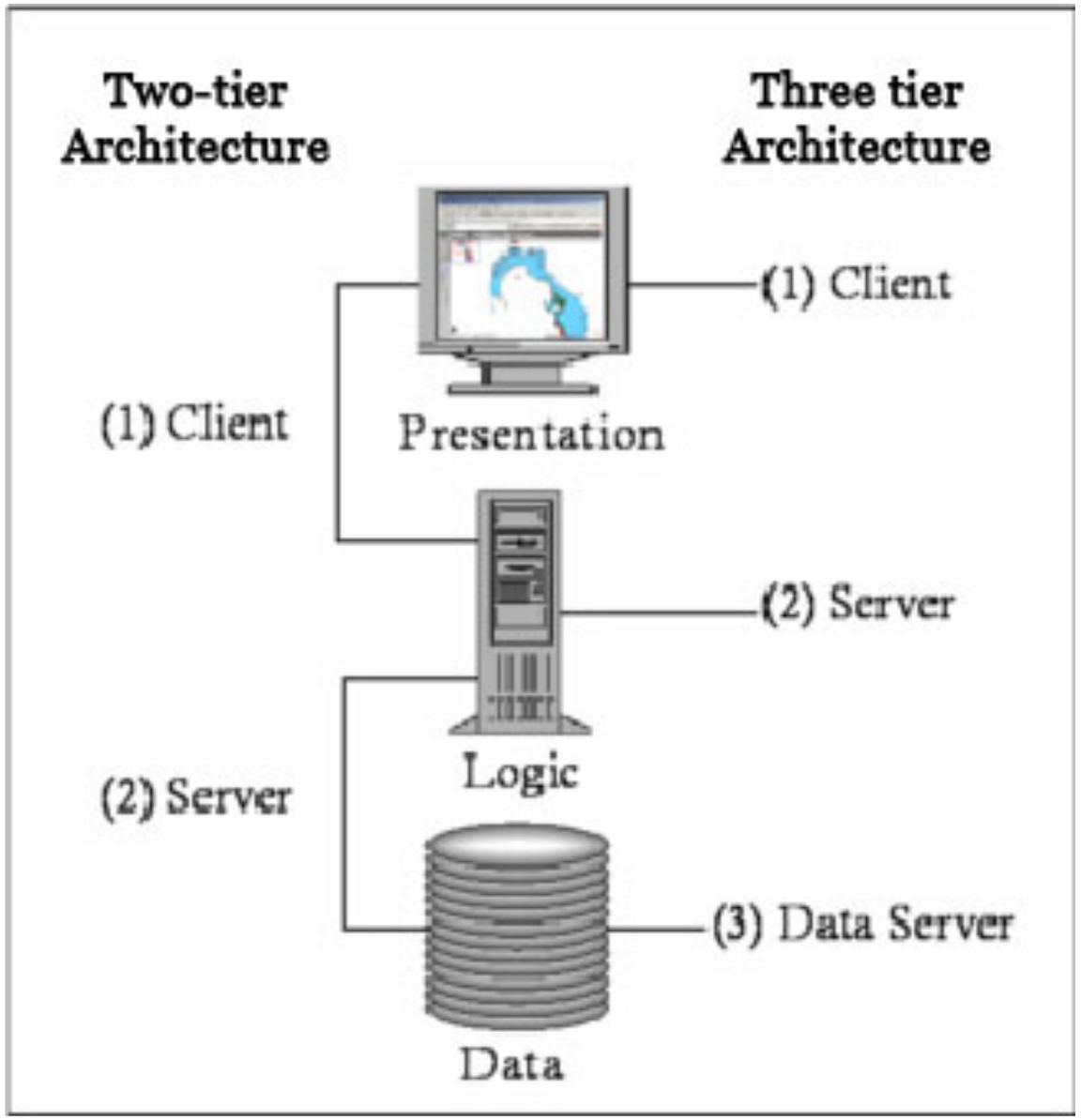
MySQL Community Server & Workbench

http://dev.mysql.com/downloads/

- Win: MySql 5.6 Installer for Windows

ColdFusion 10 Web Services

- Types of Web Services Supported
  - WSDL/SOAP
  - AJAX/JSON(P)
  - REST
  - Web Socket
- Critical to support a true 3-tiered architecture



ColdFusion 10 Web Services

- Types of Web Services Supported
  - WSDL/SOAP
  - **AJAX/JSON(P)**
  - **REST**
  - **Web Socket**

Focus of today's discussions

AJAX/JSON(P) Service

- "The Old Ways"
- Generate JSON/XML output for consumption by rich clients
- Packaged as remote CFC methods

```
<cfcomponent>

 <cffunction name="getData" access="remote" returntype="array" output="false"
           returnformat="json">
 </cffunction>

</cfcomponent>
```

Invoke via XMLHttpRequest: /mycomponent.cfc?method=getData

Converting Queries to Arrays of Structs

```
<cffunction name="query2array" access="private"
            returntype="array" output="false">

  <cfargument name="qdata" type="query" required="yes">

  <cfset local.i = "0">
  <cfset local.stdata = structnew()>
  <cfset local.thiscolumn = "">
  <cfset local.aresult = arraynew(1)>

  <cfloop from="1" to="#qdata.recordcount#" index="i">
      <cfset local.stdata = structnew()>
      <cfloop list="#qdata.columnlist#"
              index="local.thiscolumn">
        <cfset stdata[lcase(local.thiscolumn)] =
                                qdata[local.thiscolumn][i]>
      </cfloop>
      <cfset local.aresult[i] = local.stdata)>
  </cfloop>

  <cfreturn aResult>
 </cffunction>
```

Using SerializeJSON

```
<cffunction name="getDataJSON"
            access="remote"
            returntype="string"
            output="false"
            returnformat="plain">

  <cfset local.q = "">
  <cfquery name="local.q">
       select firstname, lastname
       from people
  </cfquery>

  <cfreturn serializejson(q,true)>
</cffunction>
```

```
{
    "ROWCOUNT": 3,
    "COLUMNS": [
        "FIRSTNAME",
        "LASTNAME"
    ],
    "DATA": {
        "FIRSTNAME": [
            "Steve",
            "David",
            "Dave"
        ],
        "LASTNAME": [
            "Drucker",
            "Gallerizzo",
            "Horan"
        ]
    }
}
```

Removing High Ascii Characters

You can programmatically deal with users who copy and paste from word into wysiwyg fields...

... or you can smite them.

Special chars (ellipsis, smartquotes, etc) can invalidate your JSON

Check out Ben Nadel's CleanHighAscii() method

Making AJAX Requests from jQuery Mobile

```
$.ajax({
    url: 'myservice.cfc?method=getdata',
    type: 'GET',
    dataType: 'json',
    error : function (){ alert('there was an error'); },
    success: function (data) {
        console.log(data);
        // debugger;
    }
});
```

Callback handlers are executed asynchronously

The Success handler receives the data as a javascript object

Use console.log() or debugger; to inspect results

Handling HTTP Request Payloads

```
<cfcomponent>
 <cffunction
    name="saveData"
    access="remote"
    returntype="string"
    returnformat="plain">

   <cfset var requestData = deserializeJson(
              toString(getHttpRequestData().content)
   )>

   <!--- data now available as requestData.fieldName --->
   <cfreturn serializeJson({success=true})>
 </cffunction>
</cfcomponent>
```

Normalizing Nulls

```
<cfcomponent>
 <cffunction
    name="saveData"
    access="remote"
    returntype="string"
    returnformat="plain">

   <cfset var requestData = deserializeJson(
              toString(getHttpRequestData().content)
   )>

   <!--- normalize foo if null --->
   <cfif not isdefined("requestdata.foo")>
     <cfset requestdata.foo = "">
   </cfif>

   <!--- data available as requestData.fieldName --->

   <!--- return success=true as json --->
   <cfreturn serializeJson({success=true})>

 </cffunction>
</cfcomponent>
```

Implementing CORS

Cross-Origin Resource Sharing

Name	Value	Entry Type
Access-Control-Allow-Credentials	true	Local
Access-Control-Allow-Headers	Origin, x-requested-with, Content-Type, Accept	Local
Access-Control-Allow-Methods	GET,POST	Local
Access-Control-Allow-Origin	*	Local
Access-Control-Max-Age	86400	Local
Access-Control-Request-Method	GET,POST,PUT,DELETE	Local
Allow	OPTIONS, TRACE, GET, HEAD, POST	Local
X-Powered-By	ASP.NET	Inherited

http://caniuse.com/cors

http://enable-cors.org

XHR2

XMLHttpRequest - the next generation

Supported by modern browsers (IE 10+)

Send/Receive binary data

Works with Blob

Enhanced form posting

Issues progress events

See: http://www.html5rocks.com/en/tutorials/file/xhr2/

Creating a JSONP Service

"JSON with Padding"

Dynamically injects <script> tags into your document

```
<cffunction name="getDataJSONP"
            access="remote"
            returntype="string"
            output="false"
            returnformat="plain">
  <cfargument name="callback" type="string" required="yes">

  <cfset local.q = "">
  <cfquery name="local.q">
       select firstname, lastname
       from people
  </cfquery>

  <cfreturn arguments.callback &
                          "(" & serializejson(q,true) & ")">
</cffunction>
```

Making JSON-P Requests from jQuery

```
var url='http://someurl/somecfcomponent.cfc?';

$.getJSON(url + 'method=somemethod&callback=?',
          function(data) {
            console.log(data);
          }
);
```

Headers Preview Response Cookies Timing

Request URL: http://webapps.figleaf.com/ftst2/components/subgenius.cfc?method=getlocationsjsonp&callback=jQuery171010779800522141159_1348367178551&_=1348367178590
Request Method: GET
Status Code: 200 OK

▼ Request Headers view source

Accept: */*
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8
Cache-Control: no-cache
Connection: keep-alive
Cookie: __ALC_CLIENT_ID=1875-10cce916a-fc08-46ec-8685-4b6d71c6f830; googtransopt=os=1; __utma=197579305.884680507.1344717807.1344935003.1344951044.8; __utmc=197579305; __utmz=197579305.1344717807.1.1.utmcsr=(direct)|utmccn=(direct)|utmcmd=(none); __utma=213379075.1488215638.1327960853.1345390153.1345463014.22; __utmc=213379075; __utmz=213379075.1344773655.20.8.utmcsr=localhost|utmccn=(referral)|utmcmd=referral|utmcct=/aaohns2/index.html; CFID=14837; CFTOKEN=27274163; CURRENTSITE=dashboard1
Host: webapps.figleaf.com
Pragma: no-cache
Referer: http://localhost:8500/cfjqm/solution/final/index.html
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_1) AppleWebKit/537.1 (KHTML, like Gecko) Chrome/21.0.1180.89 Safari/537.1

▼ Query String Parameters view URL encoded

method: getlocationsjsonp
callback: jQuery171010779800522141159_1348367178551
_: 1348367178590

▼ Response Headers view source

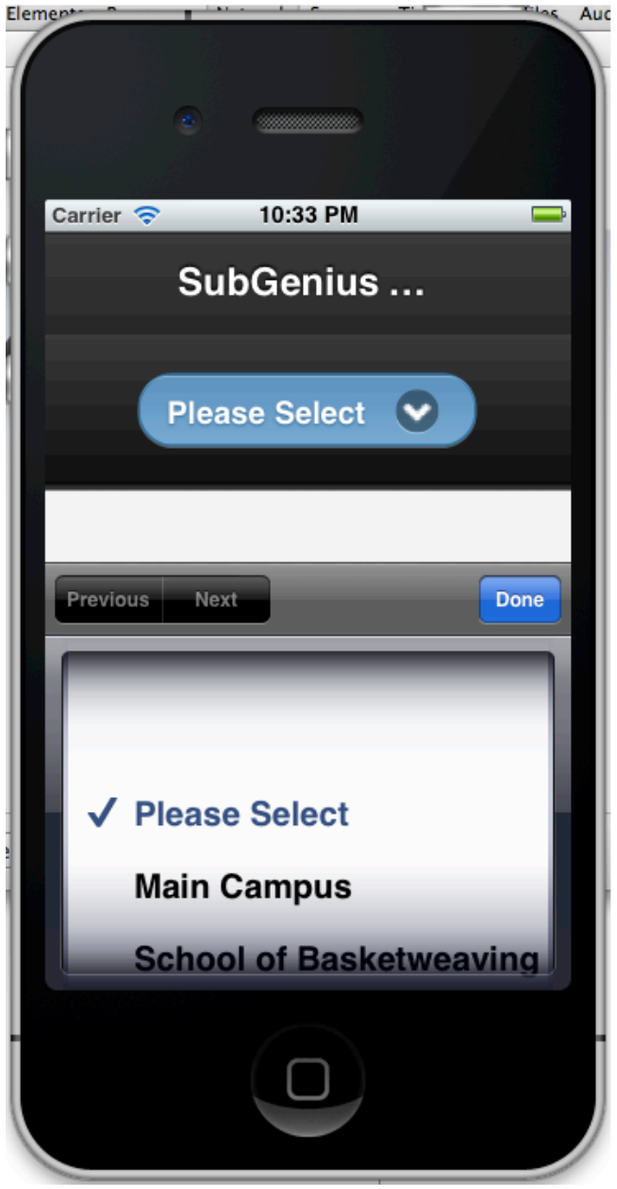
Connection: close
Content-Type: text/html; charset=UTF-8
Date: Sun, 23 Sep 2012 02:26:34 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET

Deferring Data Requests

```
$('#page2').live("pagecreate", function() {
  // execute code when page 2 gets instantiated
}
```

Walkthrough 9-1: Creating a Simple AJAX Web Service

- Creating the web service
- Testing via URL
- Reviewing data via browser debugger



REST(ful) Web Services

- Short for “Representational State Transfer”
- Provides a simple, structured method for calling web resources based on the standard URI methodology.
- Provides a consistent interface for working with data via GET, PUT, POST, and DELETE function.
- Stateless transactions, consistent with HTTP/S standards. Applications handle state management.

ColdFusion REST Services

- Supports all GET, PUT, POST, and DELETE functions as methods within a CFC
- Natively supports JSON and XML serialization/deserialization, over both HTTP or HTTPS protocols.
- As these are CFC based, the same component can be published as a REST service or a WSDL service.

Analyzing a REST Service Call

http://server /rest/ aservicepath/ cfccall

Hostname

Required reserved word

Service Path as defined in the ColdFusion Administrator

restPath value from the cfcomponent definition

Analyzing a REST Service Call

http://localhost /rest/ crimeapp/crime.json

Hostname

Required reserved word

Service Path as defined in the ColdFusion Administrator

restPath value from the cfcomponent definition

Sample REST calls

http://localhost/rest/crimeapp/crimes.json

http://localhost/rest/crimeapp/crimes.xml

http://localhost/rest/crimeapp/crimes/1.json

- method=get
- method=put
- method=delete
- method=post

REST Services Additions

- ColdFusion 10 supports the four main functions via a new <cffunction> `httpmethod` attribute: PUT (create), GET (read), POST (update), DELETE (delete).

- All REST CFCs must be in one directory; nested REST services are not supported.

Creating a REST Service CFC

```
<cfcomponent
      rest="true"
      restpath="person"
      extends="base">

  <cffunction name="GetPeople"
          access="remote"
          returntype="array"
          httpmethod="GET">

    <cfquery name="local.q">
      select * from person
    </cfquery>

    <cfreturn query2array(local.q)>
  </cffunction>

</cfcomponent>
```

Invoking a REST Service CFC

Using the previous example, we could invoke the REST service two ways.

- To generate a JSON response:

http://servername/rest/MyApp/person.json

- To generate an XML response:

http://servername/rest/MyApp/person.xml

Resource Functions

- Any CFC function that does not contain a `restPath` parameter, is considered a **Resource Function**.
- These are generally private functions used internally to support the main REST methods of service component.

Defining a GET Method

```
<cffunction
  name="getPeson"
  access="remote"
  returntype="struct"
  httpMethod="get"
  restpath="{personId}">

<cfargument name="personId"
        required="true"
        restargsource="Path"
        type="numeric" />

  <cfquery name="local.q">
    select *
    from person
    where personId = <cfqueryparam
              cfsqltype="cf_sql_numeric"
              value="#arguments.personId#">
  </cfquery>

  <cfset local.result = query2array(local.q)>

  <cfreturn local.result[1]>

</cffunction>
```

- GET method is used to read data records.
- `httpMethod` attribute is defined as "**get**"
- The **restPath** attribute defined how the arguments are expected to be presented in the URL.
- The **restargsource** attribute of the cfargument tag defines where to look for the value of the argument.
- Example shown assumes that the CFC inludes a method query2array().

Invoking a GET Method

http://servername/rest/MyApp/1.json

- In the above URL, 1 corresponds to the value of the argument **personId** in the code, e.g.

```
<cffunction
   name="getPeson"
   access="remote"
   returntype="struct"
   httpMethod="get"
   restpath="{personId}">
```

Defining a POST Method

```
 <cffunction
  name="createPerson"
  access="remote"
  returntype="numeric"
  httpMethod="post">

<cfargument name="firstname"
          required="true"
          type="string"
          restargsource="Form" />

<cfargument name="lastname"
          required="true"
          type="string"
          restargsource="Form" />

 <cftransaction>
   <cfquery>
    insert into person (firstname, lastname)
    values (
     <cfqueryparam cfsqltype="cf_sql_varchar"
                 value="#arguments.firstname#">
     <cfqueryparam cfsqltype="cf_sql_varchar"
                 value="#arguments.lastname#">
    )
   </cfquery>
   <cfquery name="local.getlast">
    select LAST_INSERT_ID() as lastid
    from person
   </cfquery>
 </cftransaction>

<cfreturn local.getlast.lastid>
```

- Used to create data records.
- No **restPath** is needed, but httpMethod is defined as "**post**" as this is where received variables are coming from.
- The **restagrsource** attribute of the cfargument tag defines where to look for the value of the argument.

Invoking a POST Method

```
  <form
action="/rest/MyApp/Person.json"
method="post">

<input type="text" name="firstname"
placeholder="First Name”>

<input type="text" name="lastname"
placeholder="Last Name”>

<input type="submit">

</form>
```

- Form action page points to the REST service URL
- Form method is POST, matching the method defined in the code for the CFC function.

Defining a PUT Method

- PUT method used to update records
- Primary key information is passed in the URL
- Field-level data is passed in POST transaction data

Defining a PUT Method

```
<cffunction
 name="updatePerson"
 access="remote"
 returntype="struct"
 httpMethod="post"
 restpath = "{personId}">

<cfargument
 name="personId"
 required="yes"
 restargsource="Path" />

 <cfargument
  name="firstname"
  required="true"
  type="string"
  restargsource="Form" />

<cfargument
  name="lastname"
  required="true"
  type="string"
  restargsource="Form" />
```

```
<cfquery>
  update person

  set firstname =
   <cfqueryparam
    cfsqltype="cf_sql_varchar"
    value="#arguments.firstname#">,

  lastname =
    <cfqueryparam
     cfsqltype="cf_sql_varchar"
     value="#arguments.lastname#">

  where personid =
    <cfqueryparam
     cfsqltype="cf_sql_numeric"
     value="#arguments.personid#">

</cfquery>

<cfreturn {personId = arguments.personId}>

</cffunction>
```

Invoking a PUT Method

```
  <form
action="/rest/MyApp/person/3"
method="post">

<input type="text" name="firstname"
value="Dave" />

<input type="text" name="lastname"
value="Watts"/>

<input type="submit"
value="Click me" />

</form>
```

- Can be invoked by a form post as shown here.
- Note the form action value, there the “3” corresponds to the primary key value for the record we want to update.

Defining a DELETE Method

```
<cffunction
  name="deleteApplicant"
  access="remote"
  returntype="struct"
  httpMethod="delete"
  restpath="{personId}">

 <cfargument
  name="personid"
  required="true"
  restargsource="Path"
  type="numeric" />

 <cfquery>
  delete
  from person
  where personid =
  <cfqueryparam cfsqltype="cf_sql_numeric"
      value="#arguments.personId#">
 </cfquery>

    <cfreturn {
    applicantId = arguments.applicantId
                operation="delete"
               }>
</cffunction>
```

- Primary key of record to delete is encoded in URL, with method set to “delete”.
- Invoked like PUT method.
- **Note: The HTTP delete method is only supported in browsers by the XMLHttpRequest object.**

Registering a REST Service

Data & Services > REST Services

Register your applications and folders to generate RESTful web services. ColdFusion automatically registers CFCs found in the registered folders.

Add/Edit REST Service

Root path /Library/WebServer/Documents/nasa/Unit9/code/ Browse Server

Application path or root folder where ColdFusion searches for CFCs

Service Mapping crimedata

Alternate string to be used for application name while calling REST service. For example, http://localhost/rest/{service mapping}/test (Optional)

Set as default application ☑

There can be only one default application (which can be changed later). Setting an application as default makes application name optional while calling the webservice. For example http://localhost/rest/path

Update Service Delete Service

Active ColdFusion REST Services

Actions	Root Path	Service Mapping	Default
	/Library/WebServer/Documents/nasa/Unit9/code/	crimedata	YES

IMPORTANT PUBLIC SAFETY TIP

Your REST CFC's must NOT be in a folder structure that contains hypens or special characters!!!!

E.g. /walk/walk10-1/myrestcfc.cfc

(re)Initializing REST Services Programmatically

```
<cfcomponent>
 <cfset this.name = "myapp">
 <cfset this.datasource="myappDSN">

<cffunction name="onApplicationStart">
 <cfset restInitApplication(
      getDirectoryFromPath(getCurrentTemplatePath()),
      this.name
  )>
 </cffunction>

 <cffunction name="onRequestStart">
    <cfif isdefined("url.init")>
       <cfset onApplicationStart()>
    </cfif>
 </cffunction>

</cfcomponent>
```

Using jQuery Mobile with REST

- You can use JQM with REST with the $.ajax() method
- or a JQM AJAX form post
- both methods require their respective parameters to be set properly to pass data as expected to the ColdFusion CFC in order to be processed.

Retrieving Data via JQM & REST

- Generally performed using the $.ajax() method
- Bulk of work is performed in the $.ajax() "success" callback function.
- In this process, the current data is erased and repopulated from the retrieved data.
- The a "refresh" method is called on the UI component after it is repopulated
- Any UI event listeners are re-bound to the controls

Retrieving Data via JQM & REST

```
var myApp = {}; // used to cache data results in global ns
  var restUrl = "/rest/cfjqm_solution/";

  $('#personView').live("pagecreate", function() {
   $.ajax({
     url: restUrl + "person.json",
     type: 'GET',
     dataType: 'json',
     error : function (){
       alert('there was an error');
       console.log(arguments)
     },
     success: function (result) {
       myApp.people = result;

       // delete list items
       $('#personList').empty();

       // loop over list items and add to list
       for (var i=0; i<result.length; i++) {
         var out = "<a href='#'>" + result[i].firstname + " " + result[i].lastname;
         $('#applicantList')
          .append($("<li></li>")
          .attr("data-value",result[i].personid)
          .html(out) );
       }
   }
 }); // ajax
```

Adding Data via JQM & REST

```
<div data-role="page" id="ContactForm">
      <div data-role="header"
            data-position="fixed"
             data-id="brandingbar">
       <h1>Add a Contact</h1>
      </div>
      <div data-role="content" id="ContactFormDetail">
       <form
           action="/rest/cfjqm_solution/applicant.json"
           method="post"
           data-ajax="true">

          <fieldset data-role="controlgroup">
           <div data-role="fieldcontain">
              <label for="firstNameField">First Name</label>
              <input type="text"
                     name="firstname"
                     id="firstNameField"
                     data-prevent-focus-zoom="true"
                     data-mini="true" />
           </div>
           <div data-role="fieldcontain">
              <label for="lastNameField">Last Name</label>
              <input type="text"
                     name="lastname"
                     id="lastNameField"
                     data-prevent-focus-zoom="true"
                     data-mini="true" />
            </div>
           </fieldset>

           <fieldset class="ui-grid-a">
           <div class="ui-block-b"
                style="float:none; text-align:center;
                       margin-left: auto; margin-right: auto">

              <button
                type="submit"
                data-theme="a">Submit</button>
           </div>
         </fieldset>
       </form>
     </div>
   </div>
```

- When adding data, you will usually POST data from a JQM form.
- Note the form action parameter points to the REST service
- The method declares this as a form POST
- data-ajax parameter declares this as a JQM AJAX POST.

Updating Data via JQM & REST

```
<script type="text/javascript">

 fillForm = function(rec) {

    var restUrl = '/rest/cfjqm_solution/applicant';

    $('#ContactFormDetail > form')
     .attr('action',restUrl + '/' + rec.personid + '.json');
    }

    // fill rest of form fields
    $('#firstNameField')
        .val(rec.firstname)
        .textinput('refresh');

   $('#lastNameField')
        .val(rec.lastname)
        .textinput('refresh');

  }

 </script>

  // code omitted for brevity
  <form
         action="/rest/cfjqm_solution/applicant.json"
         method="post"
         data-ajax="true">


  </form>
  // code omitted for brevity
```

- The value of the “type” attribute is “delete”
- The primary key of the record to delete is built into the URL, and appended with the “.json” extension.
- **Note: You should always secure your REST services using ColdFusion's <cflogin> framework and the <cffunction> ROLES attribute.**

Deleting Data via JQM & REST

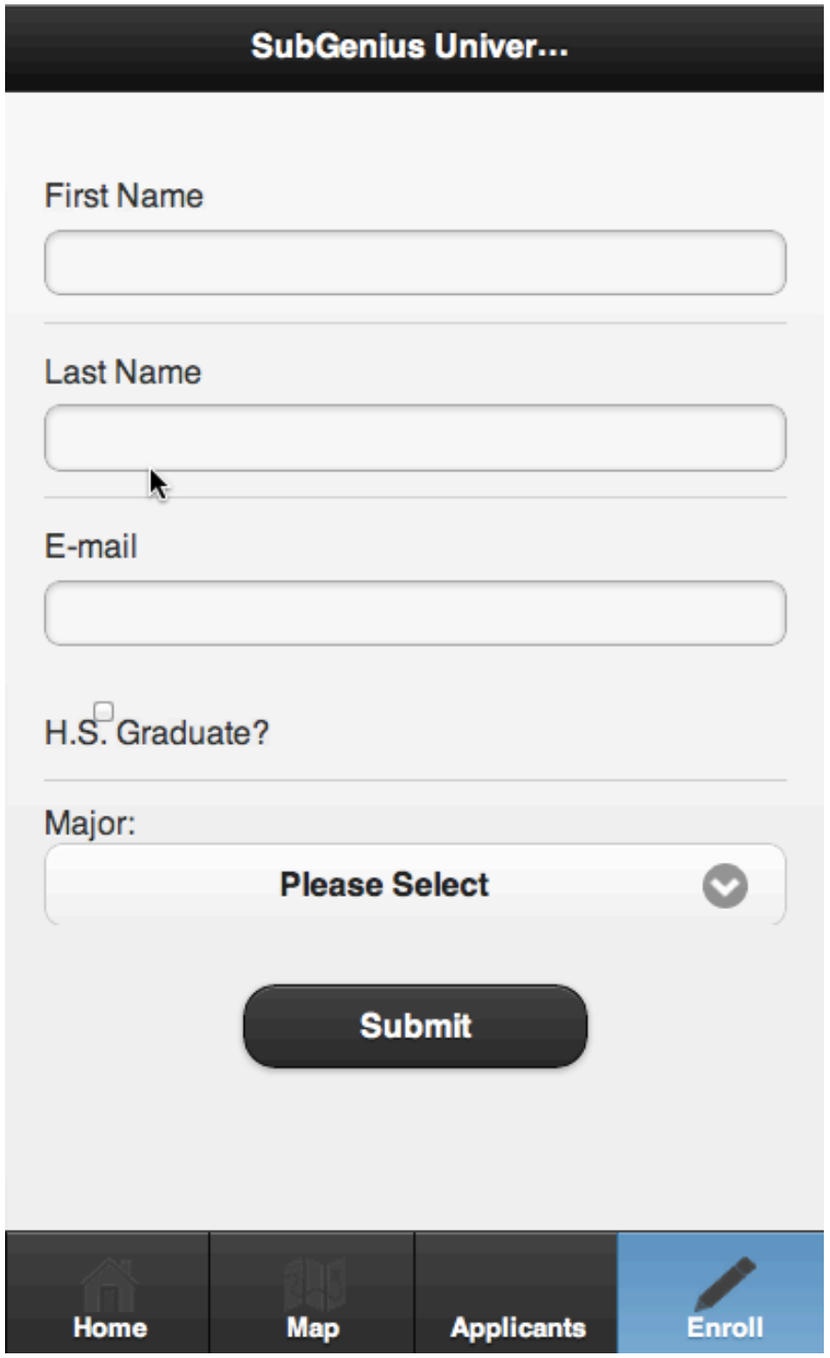
```
deleteRecord = function(id) {
  var restUrl = "/rest/
cfjqm_solution/person/";

   $.ajax({
   url: restUrl + id + '.json',
    type: 'delete',
    dataType: 'json',
    error : function (){
      alert('there was an error');
      console.log(arguments)
    },
    success: function (result) {
     alert("Record Deleted");
    }
  }); // ajax
}
```

- The value of the “type” attribute is “delete”
- The primary key of the record to delete is built into the URL, and appended with the “.json” extension.
- **Note: You should always secure your REST services using ColdFusion's <cflogin> framework and the <cffunction> ROLES attribute.**

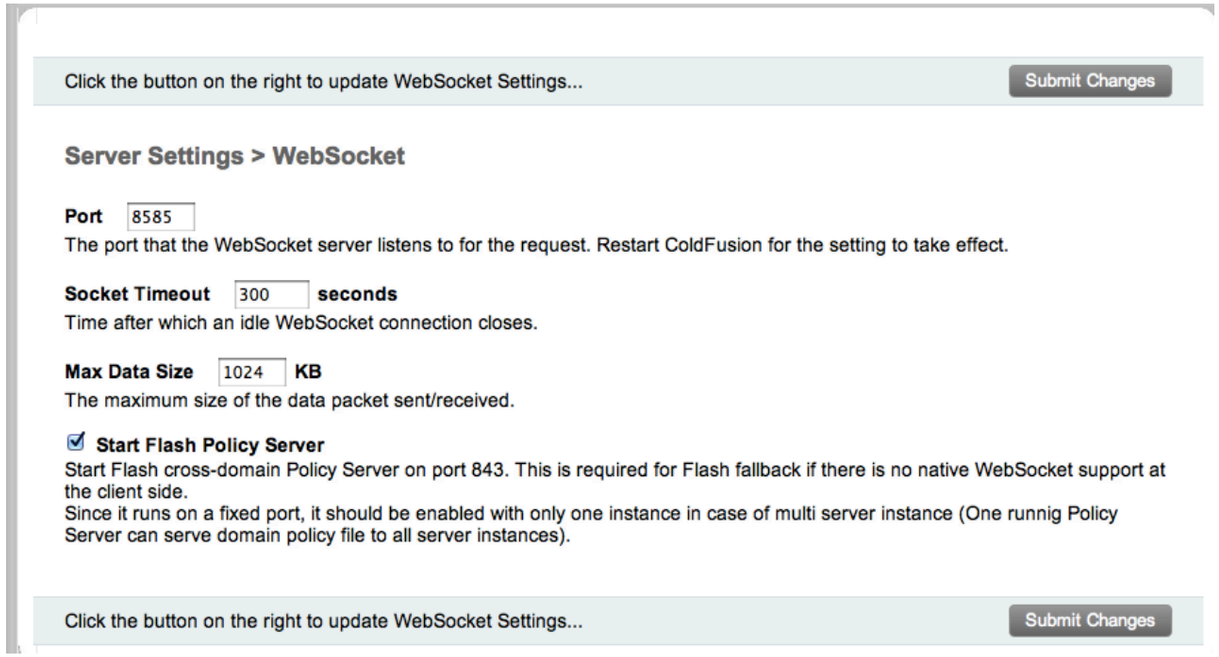
Walkthrough 9-2: Creating a REST Web Service

- Create a REST service CFC
- Registering in CF Administrator
- Populating a JQM listview with data retrieved from a REST service
- POSTing form data via JQM to a REST service



HTML5 Web Sockets

- Full-duplex communications
- Bi-directional over single TCP socket
- Persistent connection between client and server



ColdFusion 10 & Web Sockets

- Configurable port via CF Admin
- Socket timeout for auto closure
- Configurable packet size
- Fallback to Flash if native HTML5 web socket support not provided by browser (and Flash is installed)
- ColdFusion tags make implementing web socket quite easy

Communication Modes

- One-to-One (P2P)
  - Simple implementation; no need for publisher/subscription based model.
- One-to-Many (Broadcast)
  - Follows Publisher/Subscriber model
  - A communications “channel” is established, and listeners “subscribe” to the channel
  - Any messages published to the channel are received by all listeners

Broadcast Models

- **Server Relay**
  In this model, messages are simply passed through as-is from publisher to subscribers.
- **Server Process & Forward**
  In this mode, the server will process incoming messages before they are sent along to any listening subscribers.
- **Server push**
  These are messages initiated by the server itself, rather than by a specific publisher.

Configuring Web Sockets

Click the button on the right to update WebSocket Settings... Submit Changes

Server Settings > WebSocket

Port 8585
The port that the WebSocket server listens to for the request. Restart ColdFusion for the setting to take effect.

Socket Timeout 300 **seconds**
Time after which an idle WebSocket connection closes.

Max Data Size 1024 **KB**
The maximum size of the data packet sent/received.

☑ **Start Flash Policy Server**
Start Flash cross-domain Policy Server on port 843. This is required for Flash fallback if there is no native WebSocket support at the client side.
Since it runs on a fixed port, it should be enabled with only one instance in case of multi server instance (One runnig Policy Server can serve domain policy file to all server instances).

Click the button on the right to update WebSocket Settings... Submit Changes

Configuring Web Sockets

Port	This is the default port over which a web socket connection will be established if not defined upon creation.
Socket Timeout	The timeout for the connection if no activity is detected. This value will depend on the types of applications running on the server.
Max Data Size	This is the default size of the data packet sent over the persistent connection. Although adjustable, the default value should be adequate for most applications.
Start Flash Policy Server	Only needed if you are going to attempt support for client that do not support native HTML5 web socket services.

Using Flash Policy Server

Situation	Actions
HTML5 native websocket support unavailable, but Flash IS installed	System falls back to Flash for communications.
HTML5 native websocket support unavailable, but Flash IS NOT installed	Sends a message indicating that the connection is not successful. To proceed, either move on to a compliant browser or install Flash.

Using Broadcast Mode

```
<cfcomponent>
  <cfset this.name = "cfjqm_solution">
  <cfset this.datasource="cfjqm">
  <cfset this.wschannels = [
        {
         name : "chat",
         cfcListener : "ChatListener"
        }
  ]>
  ...
</cfcomponent>
```

- One-to-many is the predominant use case for this mode.
- Communications are performed over defined “channels”
- Channels are defined in Application.cfc.
- “subchannels” like chat.tech can be created after initial channel is defined.

Defining Channels

Unit9/websocket/Application.cfc

```
component
{
   this.name = "MyApp";
   this.wschannels =
      [
       {name : "weather"},
       {name : "chat")
      ];
}
```

<cfwebsocket> tag

```
<cfwebsocket
  name="websocketName"
  onMessage="JavaScript function name"
  onOpen="JavaScript function name"
  onClose="JavaScript function name"
  onError="JavaScript function name"
  useCfAuth=true|false
  subscribeTo="channel_list">
```

Attribute	Description
name	Name of the socket connection used by all referring JavaScript functions.
onMessage	Function to call upon receipt of a message on the listening channel(s).
onOpen	Function to call upon establishing a new web socket connection.
onClose	Function to call upon closing an established web socket connection.
onError	Function to call upon the detection of an error in connecting to a web socket.
useCfAuth	If **true** (default value), user is assumed to be authenticated. If false, user will need to provide authentication to proceed with using the socket connection.
subscribeTo	A comma-delimited list of the named web socket channels this connection should listen on. These must list names established in the Application.cfc file.

<cfwebsocket> Example

```
<cfwebsocket
    name = "weatherSocket"
    onOpen = "startWeatherAlerts"
    onMessage = "showWeatherAlert"
    onClose = "closeWeatherAlerts"
    subscribeTo = "weather" />

<script>
 var startWeatherAlerts = function(){
    alert('Starting weather monitoring');
 }

 var closeWeatherAlerts = function(){
    alert('Ending weather monitoring');
 }

 // 'alert' is object passed in from from received message
 var showWeatherAlert = function(alert){
    messageTxt = alert.data;
    if((messageTxt != '') && (alert.type == 'data')){
        alert('Weather alert: ' + messageTxt);
    }
 }
</script>
```

Handling Message Responses

```
▼ Object
    clientid: 91754811
    code: 0
    msg: "ok"
    ns: "coldfusion.websocket.channels"
    reqType: "welcome"
    type: "response"
  ▶ __proto__: Object
▼ Object
    channelssubscribedto: "alerts"
    clientid: 91754811
    code: 0
    msg: "ok"
    ns: "coldfusion.websocket.channels"
    reqType: "subscribeTo"
    type: "response"
  ▶ __proto__: Object
```

- Initial call will generate a "welcome" message and a "subscribeTo" message.
- Because you are listening and publishing over the same "channel", when you send a message you will get 2 responses:

```
▼ Object
    clientid: 91754811
    code: 0
    msg: "ok"
    ns: "coldfusion.websocket.channels"
    reqType: "publish"
    type: "response"
  ▶ __proto__: Object
▼ Object
    channelname: "alerts"
    data: "Major Storm Approaching!"
    ns: "coldfusion.websocket.channels"
    publisherid: 91754811
    type: "data"
  ▶ __proto__: Object
>
```

(1) the acknowledgement of the send

(2) the message that was published over the channel

JWM and CF 10 Web Sockets

- Because you are on a mobile device, you cannot using the <cfwebsocket> tag is problematic
- You can use JavaScript files that ship with CF 10, but they must be loaded manually
- Once loaded, you can then setup event listeners to leverage the objects/methods defined within these files.

Loading CF 10 Web Socket Files

Step one is to load the files from your remote server:

```
<script type="text/javascript">
 _cf_loadingtexthtml="<img alt=' ' src='/CFIDE/scripts/ajax/
 resources/cf/images/loading.gif'/>";

 _cf_contextpath="";
 _cf_ajaxscriptsrc="http://localhost:8500/CFIDE/scripts/ajax";
_cf_jsonprefix='//';
_cf_websocket_port=8575;
_cf_flash_policy_port=1243;
</script>
```

Loading CF 10 Web Socket Files

Step two is to add <script> tags to load ColdFusion's undocumented websocket Javascript API from the following URLs:

```
http://localhost:8500/CFIDE/scripts/ajax/messages/cfmessage.js
http://localhost:8500/CFIDE/scripts/ajax/package/cfajax.js
http://localhost:8500/CFIDE/scripts/ajax/package/cfwebsocketCore.js
http://localhost:8500/CFIDE/scripts/ajax/package/
cfwebsocketChannel.js
```

Defining Event Listeners

See Pages 45-46

Walkthrough 9-3: Creating a Simple Web Socket Application

- Define a Web Socket channel
- Define and initialize a Web Socket connection between your JQM app and ColdFusion 10.
- Send and receive data from a Web Socket

Unit Summary

- ColdFusion 10 provides web services support for JQM via standard CFC.
- Supported formats best used with JQM include AJAX/JSON(P) and REST.
- HTML5 Web Sockets are supported, but client-side <cfwebsocket> tag cannot be used with mobile applications
- Web Sockets provide a new, persistent connection mechanism for web applications.