

Comp 422 - Software Development for Wireless and Mobile Devices

Fall Semester 2016 - Week 5

Dr Nick Hayward

Contents

- DEV week update
- initial tests and tools
 - *a few additional testing options*
- API plugin examples
 - *geolocation*
 - *file - read, write...*
- data considerations
- local storage
- extra notes
 - *Design mockups and prototypes*

DEV week overview...

- begin development of a mobile application using Apache Cordova
 - *not a responsive website viewed on a mobile device*
- can be platform agnostic (cross-platform) or specific targeted OS
 - *eg: Android, iOS...using Cordova APIs*
 - *consider choice, and explain why?*
- outline concept, research conducted to date
- consider applicable design patterns
- are you using any sensors etc?
 - *how, why?*
- prototyping
 - *demo current prototypes*
 - *any working tests or models etc*

DEV week presentation and demo...

- brief presentation or demonstration of current project work
- ~ 5 to 10 minutes per group
- analysis of work conducted so far
 - *eg: during semester & DEV week*
- presentation, demonstration, or video overview...
 - *outline mobile app*
 - *show prototypes and designs*
 - *explain what works & does not work*
 - ...

**Scheduled for Friday 14th October
2016 @ 2.45pm**

Cordova app - test with local tools

- default testing options with Cordova CLI include
 - *emulate and run*
- other options available as well
- Apache Ripple - install using NPM

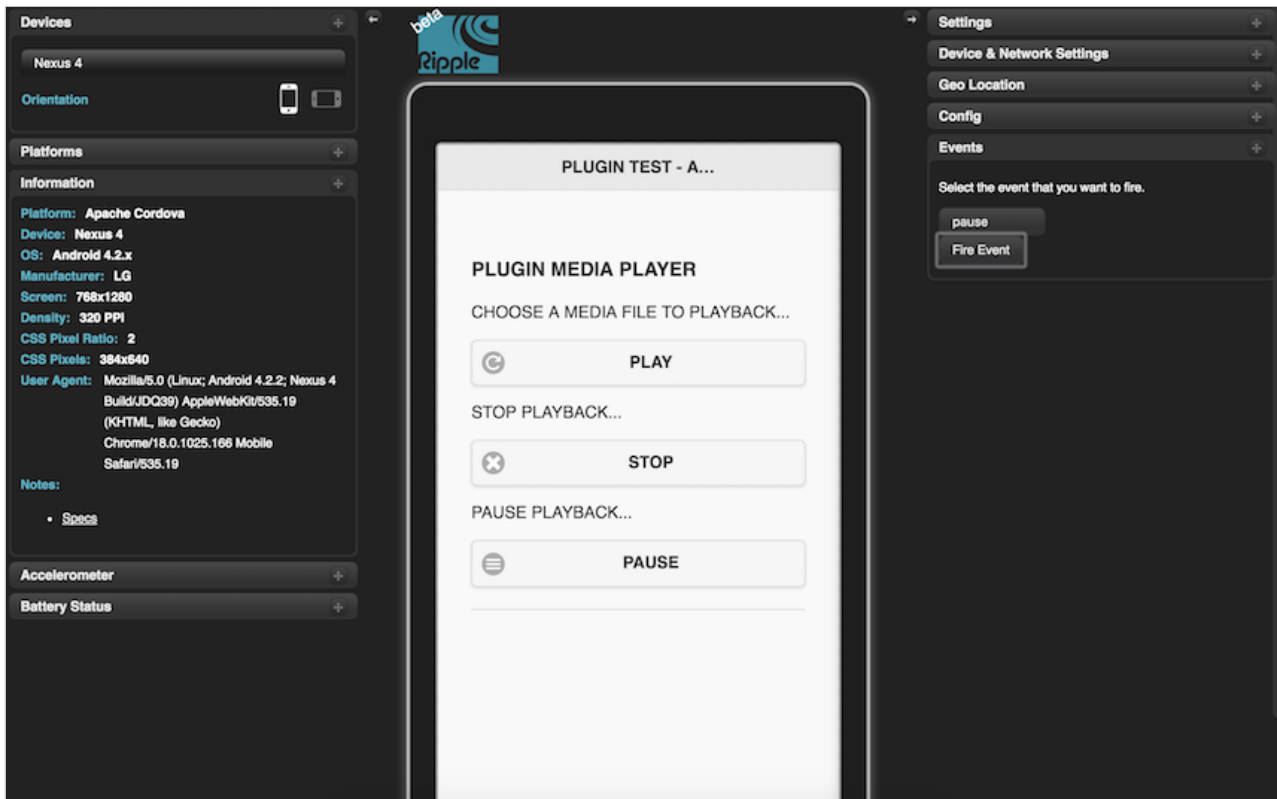
```
npm install -g ripple-emulator
```

- then, cd to working directory of Cordova app and run the following command

```
ripple emulate
```

- not the most up to date emulator, but useful for quick UI and interaction testing
- Genymotion - target at Android development, testing, and provision
 - *professional development and testing options available*
 - *further details at <https://www.genymotion.com>*

Image - Cordova app - test with local tools - Apache Ripple



Cordova app - test with local tools - Apache Ripple

Cordova app - test with local tools - serve

- Cordova also provides the option to **serve** a current app
- `serve` as self-hosted site for testing

```
cordova serve
```

- start a local static file server at `http://localhost:8000`
 - *then navigate to a given platform's directory*
 - *and the associated project UI and build*
 - *useful for UI testing and quick development*

Image - Cordova app - test with local server - serve

Package Metadata

| | |
|-------------|------------------------|
| name | Plugin Test 0.2 |
| packageName | com.example.pluginTest |
| version | 0.0.2 |

Platforms

- [ios](#)
- *osx*
- [android](#)
- *ubuntu*
- *amazon-fireos*
- *wp8*
- *blackberry10*
- *www*
- *firefoxos*
- *windows*
- *webos*
- [browser](#)

Plugins

- cordova-plugin-compat
- cordova-plugin-device
- cordova-plugin-file
- cordova-plugin-media
- cordova-plugin-whitelist

Cordova app - test with local server - serve

Cordova app - test with local tools - Chrome browser and device

- test and develop Android applications with **devices** on Chrome browser
- after running our app on a connected device, e.g.

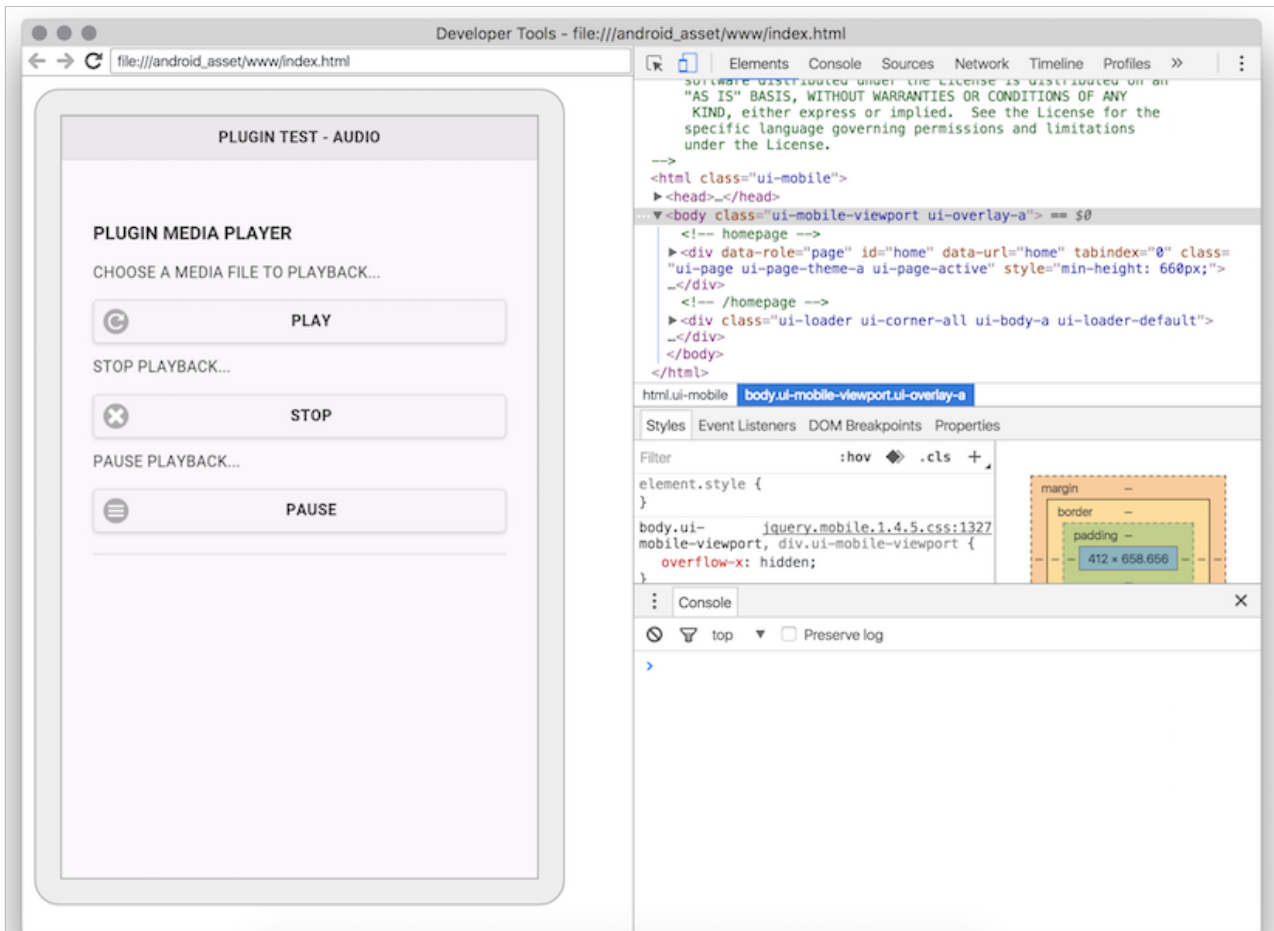
```
cordova run android
```

- inspect the app using Chrome's developer tools at the following URL,

```
chrome://inspect/#devices
```

- then select the option to `inspect` a connected device
- shows window with the standard Chrome developer tools and options
 - *inspect the DOM, JS console, styles, and so on...*
 - *use inspect option to control, navigate, and interact with our running app*

Image - Cordova app - test with local server - Chrome



Cordova app - test with local server - Chrome

Cordova app - test with Browser platform

- Cordova recently added a **Browser** platform option
- use to create a quasi-test environment for our apps
- install browser support as a standard platform

```
cordova platform add browser
```

- load our app into the browser using the following command,

```
cordova run browser
```

- platform will be useful for testing UI design and development
- many of the plugins are supported as well
 - e.g. *camera*

n.b. other options better for testing development of custom or OS level Android or iOS features...

Image - Cordova app - test with browser platform



Cordova app - browser platform

Cordova app - API plugin examples - plugin test 3

plugins - geolocation

- add and use Cordova's **Geolocation** plugin
- helps us provide information about current location of user's device
- plugin returns data on device's location
 - *including latitude and longitude*
- plugin can use the following to help determine location
 - *GPS, network signals, phone network IDs...*
- API has been developed around the W3C's **Geolocation API Specification**
- **n.b.** may not always be able to return a reliable location due to
 - *location restrictions*
 - *lack of access to a network*
 - *a user may reject location tracking and awareness...*
- need to be aware of potential privacy and security concerns
 - *application's privacy policy important*
 - *how we collect and whether we store data or not*
 - *how and when we share such data with 3rd-party services*
- consider offering user a simple opt-in/out option for location services
 - *app needs fallback options to cover lack of location services*

Cordova app - API plugin examples - plugin test 3

plugins - geolocation

- now create our test application for the **geolocation** plugin

```
cordova create pluginTestGeo com.example.pluginTest3 PluginTestGeo
```

- add our required platforms for support and development,

```
cordova platform add android --save
```

- update the www directory, modify initial settings in `config.xml`, and run initial test

```
//test in the Android emulator
cordova emulate android

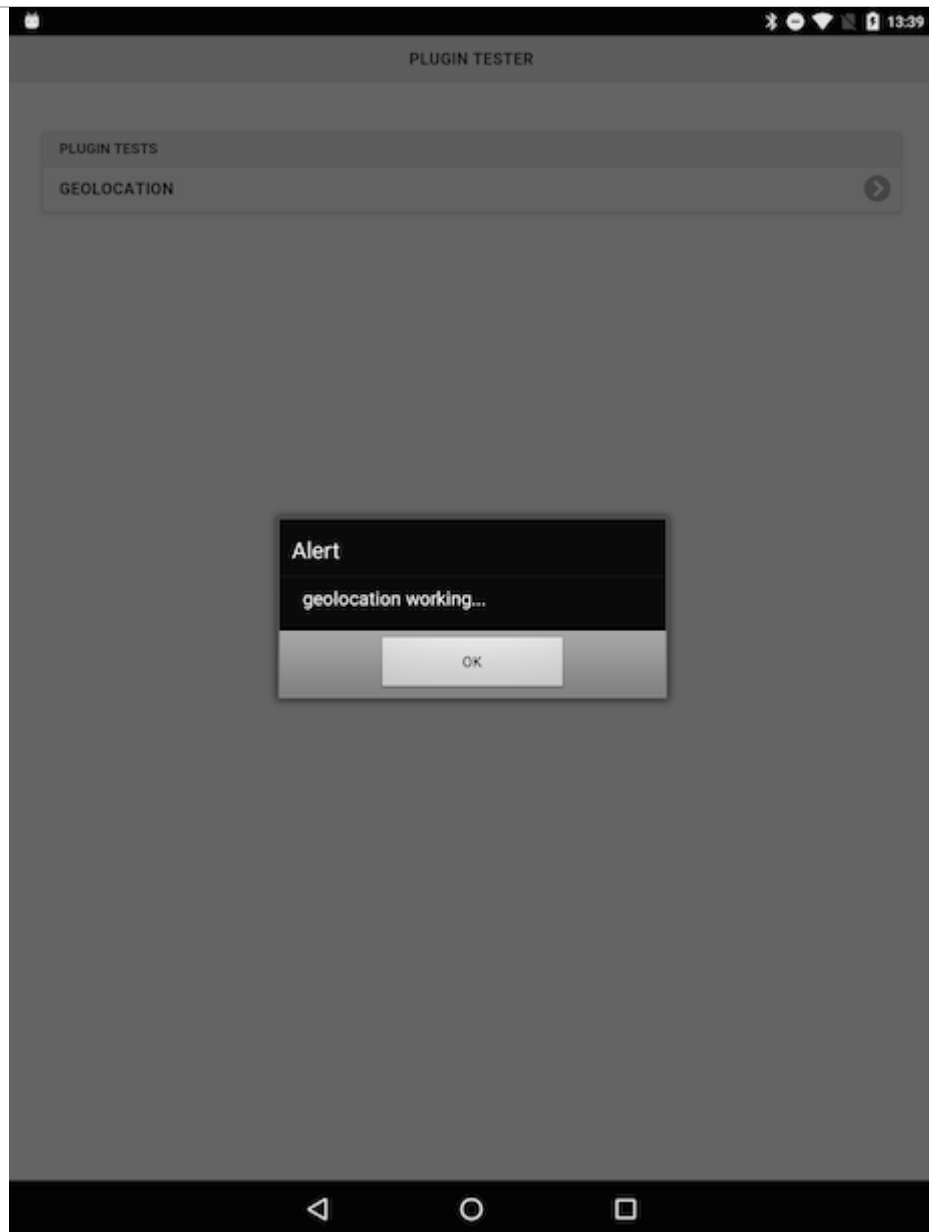
//test on a connected Android device
cordova run android
```

- add **geolocation** plugin to our new project using the Cordova CLI

```
//cordova version 5.0+
cordova plugin add cordova-plugin-geolocation

//install directly via repo url
cordova plugin add https://github.com/apache/cordova-plugin-geolocation.git
```

Image - API Plugin Tester - Geolocation



API Plugin Tester - geolocation up and running

Cordova app - API plugin examples - plugin test 3

plugins - geolocation - test plugin

- add option to check and return current location of the user's device
- add a button to allow the user to request their current location
 - *then get the location's latitude and longitude*
 - *then output the location results to the user*

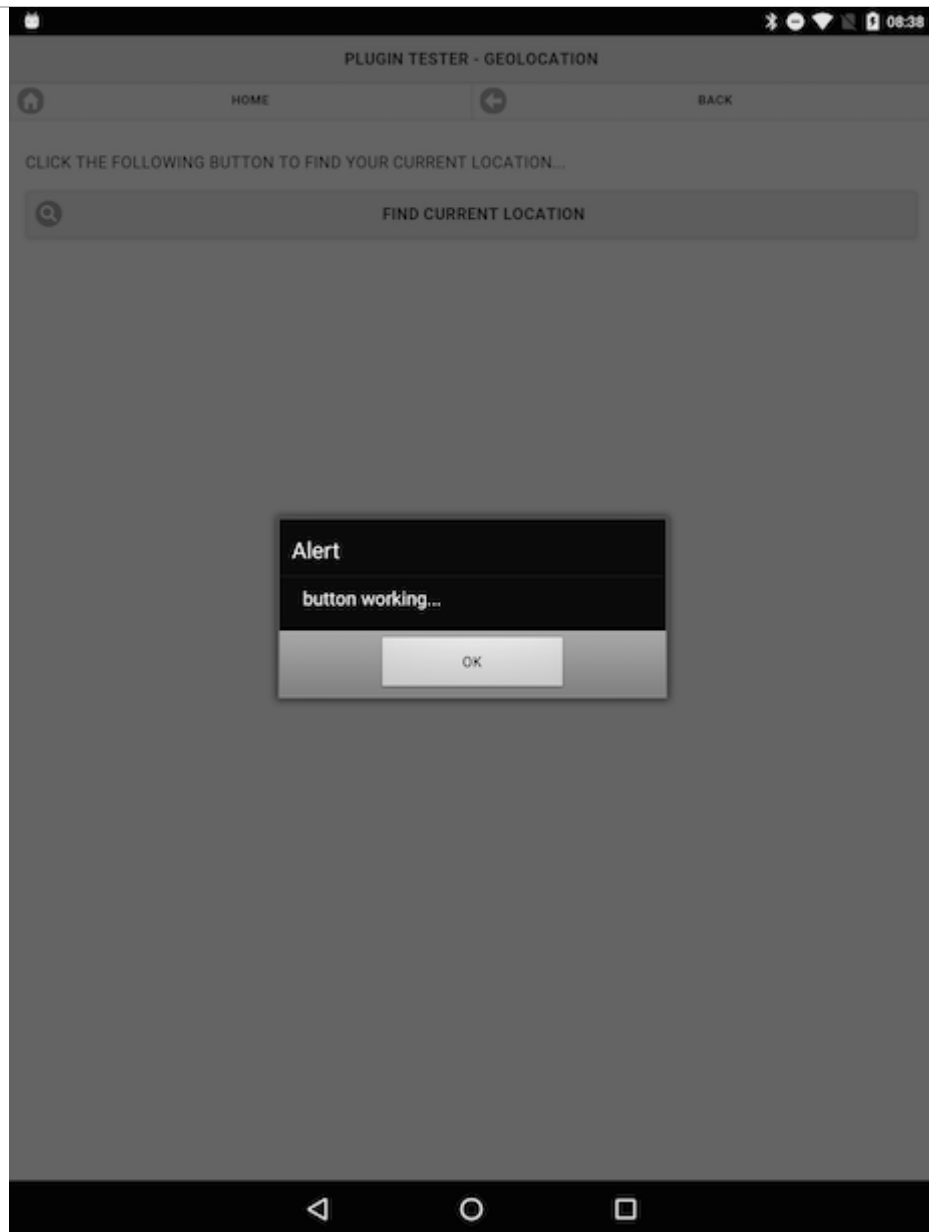
```
<div data-role="content">
  <p>Click the following button to find your current location...</p>
  <input type="button" id="getLocation" data-icon="search" value="Find Current Location" />
</div>
```

- then update the `plugin.js` file to handle the tap event for this button

```
//handle button press for geolocation
$("#getLocation").on("tap", function(e) {
  e.preventDefault();
  alert("button working...");
})
```

- output test alert for handler

Image - API Plugin Tester - Geolocation



API Plugin Tester - test geolocation button

Cordova app - API plugin examples - plugin test 3

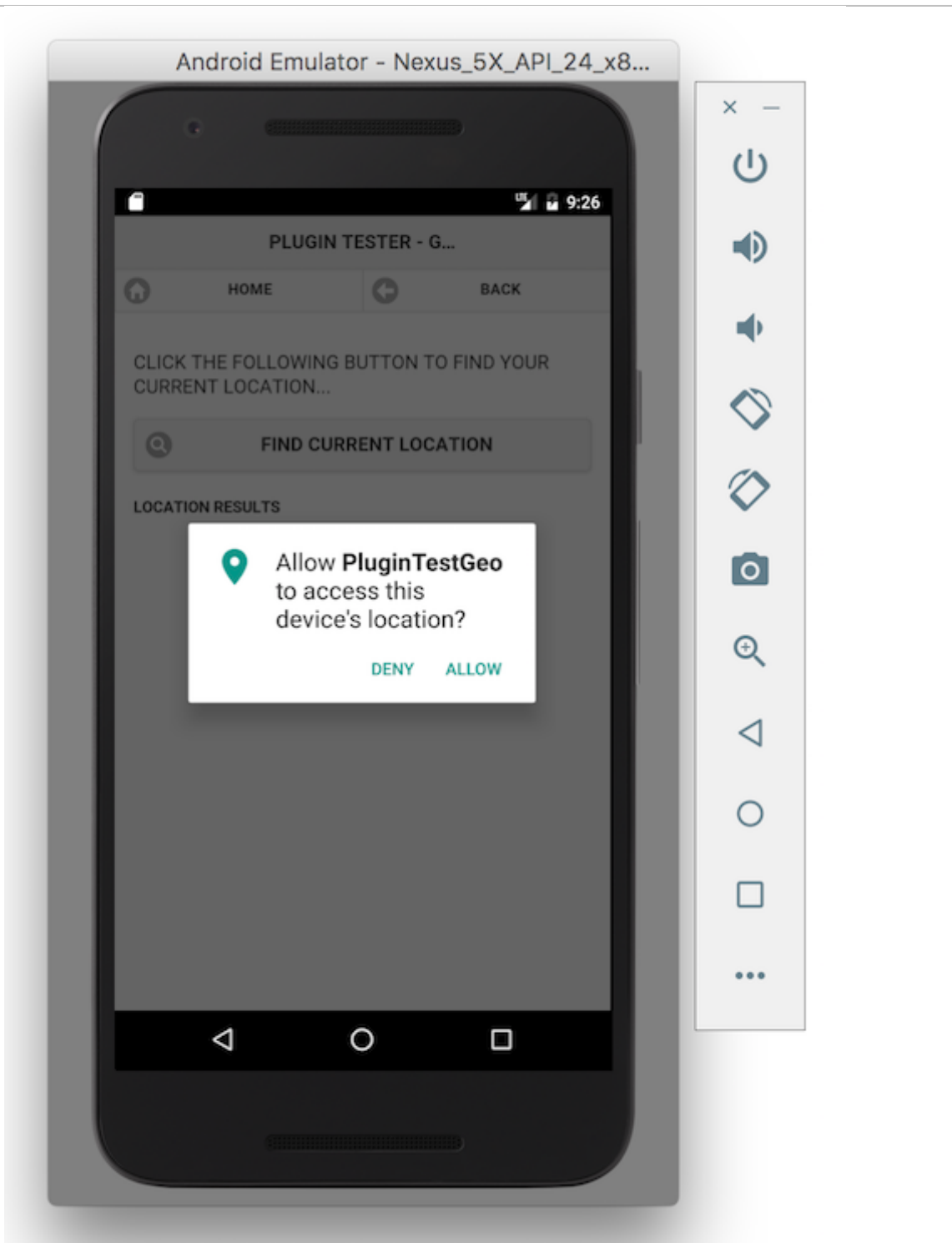
plugins - geolocation - test plugin

- add our logic for working with the navigator object and the geolocation plugin
- first function we need to add is `getLocation()`
 - *use navigator object to get current position of user's device*
- add our standard success and fail callbacks
 - *initially add a timeout for poor signal or reception*
 - *enable high accuracy for this check*
 - *asking plugin to use most accurate source available, e.g. GPS*
- `getLocation()` function is as follows,

```
function getLocation() {  
    navigator.geolocation.getCurrentPosition(onSuccess,  
        onFail, {  
            timeout: 15000,  
            enableHighAccuracy: true  
        });  
}
```

- standard callbacks for `onSuccess` and `onFail`

Image - API Plugin Tester - Geolocation permissions



API Plugin Tester - check geolocation permissions

Cordova app - API plugin examples - plugin test 3

plugins - geolocation - test plugin

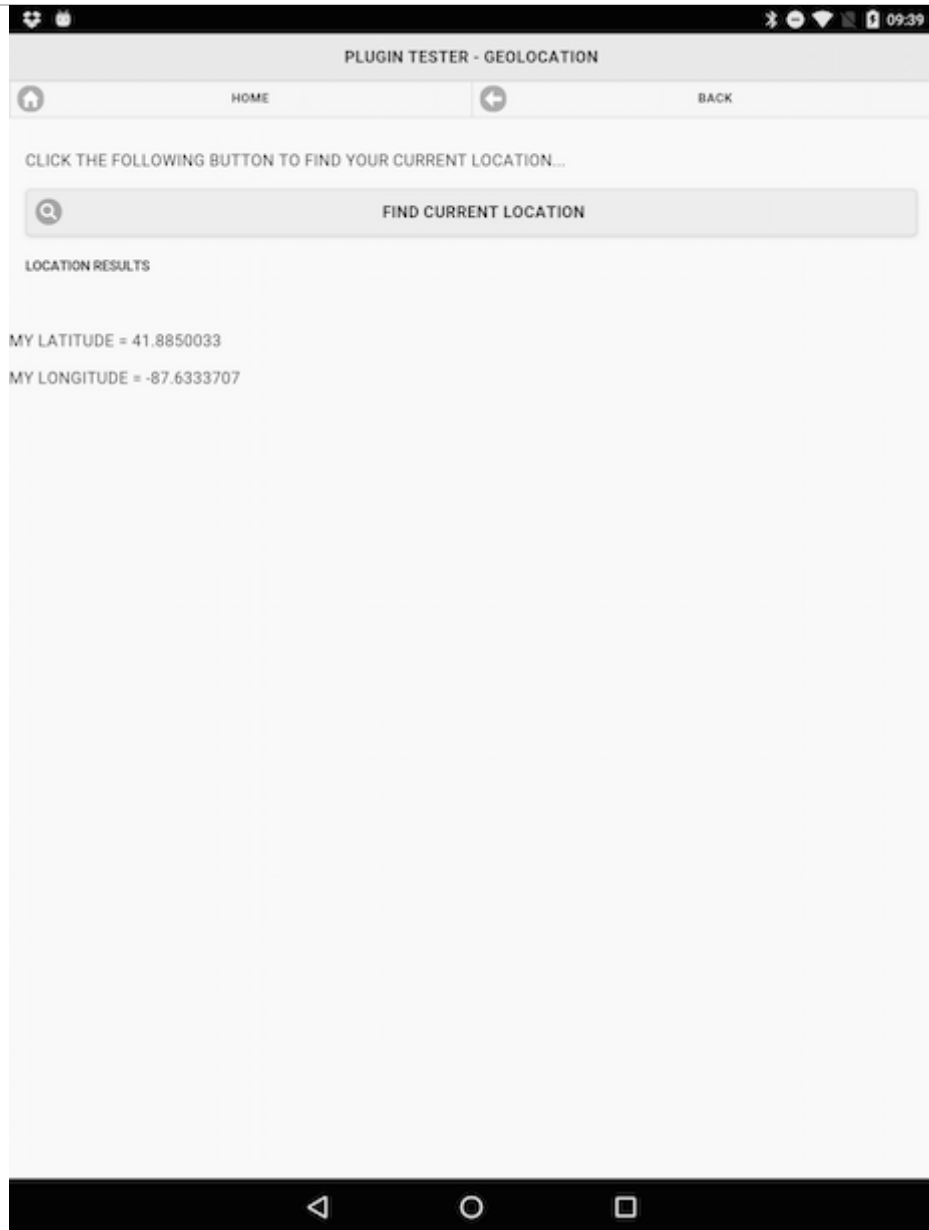
- successful return of location data
 - *use the latitude and longitude coordinates within our application*

```
function onSuccess(location) {  
    var myLatitude = location.coords.latitude;  
    var myLongitude = location.coords.longitude;  
    //output result to #location div...  
    $("#location").append("<p>my latitude = "+myLatitude+"</p><p>my longitude = "+myLongitude+"</p>");  
}
```

- now store coordinates of user's location as latitude and longitude values
- various options for usage per application
 - *render to page, use with maps, add metadata to photos, track navigation...*
- also need to allow for the possibility of errors
 - *set our onFail callback as follows*

```
function onFail(error) {  
    $("#location").append("location error code = "+error.code+" message = "+error.message);  
}
```

Image - API Plugin Tester - Geolocation



API Plugin Tester - geolocation output

Cordova app - API plugin examples - plugin test 3

plugins - geolocation - plugin options

- additional options and properties available to us in the callbacks
 - *navigator object and properties for returned `location` object*
- add options to navigator object for geolocation
 - *`maximumAge` - cached position as long as it is not older than the specified age*
 - *age is specified as a number in milliseconds, e.g. `maximumAge: 3000`*
- returned `location` object properties
 - **`altitude`** - *`location.coords.altitude`*
 - **`heading`** - *`location.coords.heading`*
 - **`speed`** - *`location.coords.speed`*
 - **`timestamp`** - *`location.timestamp`*
- fine-tune results for our users

Cordova app - API plugin examples - plugin test 3

plugins - geolocation - monitor location

- set plugin to monitor a device's location for changes

```
navigator.geolocation.watchPosition
```

- checking user's device for changes in their current location
 - *then returns device's location if a change is detected*

```
var watchID = navigator.geolocation.watchPosition(onSuccess, onFail,  
{option...}  
);
```

- error callback and options are both optional
- also use returned ID with a `clearWatch()` function to stop ongoing location check and monitoring

Cordova app - API plugin examples - plugin test 3

plugins - geolocation - manual toggle

- add a toggle option to allow a user to choose
 - *auto or manual refresh of their location*
- toggle set to **on** - app will **watch** a user's position
- toggle set to **off** - explicit location request required
- option to disable `watchPosition()` helps save battery life, reduces privacy issues...
- toggle switch initially set to default **off** position
 - *location position requires explicit request*
- toggle switch set to **on**
 - *app calls `watchPosition()` method against global `navigator.geolocation` object*

Cordova app - API plugin examples - plugin test 3

plugins - geolocation - manual toggle

- add a toggle switch to our UI

```
<form>
  <label for="flip-select">watch location:</label>
  <select id="setWatch" name="flipWatch" data-role="flipswitch">
    <option>off</option>
    <option>on</option>
  </select>
</form>
```

- then update our JS logic to handle a UI event on this widget

```
$("#setWatch").on("change", function(e) {
  e.preventDefault();
  $watchState = $(this).val();
  console.log("watch state is now set to "+$watchState);
});
```

- add a check for the current value of the toggle switch
 - *add to a property \$watchState*
 - *simply checking set value of option for the switch*

Cordova app - API plugin examples - plugin test 3

plugins - geolocation - manual toggle

- as a user **changes** the state of the toggle switch to on
 - *need to call `watchPosition()` method*
 - *start constant polling of geolocation object*

```
$("#setWatch").on("change", function(e) {  
    e.preventDefault();  
    //get state of toggle  
    $watchState = $(this).val();  
    //output check of toggle state  
    console.log("watch state is now set to "+$watchState);  
    //check state of toggle  
    if ($watchState === "on") {  
        //call function to start watching...  
        getWatchID();  
        //output check of watchID  
        console.log("watchID = "+watchID);  
    }  
});
```

- add a new function `getWatchID()`
 - *allows us to set a value for a `watchID` property*
 - *property set against `onDeviceReady()` function*

```
function getWatchID() {  
    watchID = navigator.geolocation.watchPosition(onSuccess,  
    onFail, {  
        enableHighAccuracy: true  
    });  
}
```

Cordova app - API plugin examples - plugin test 3

plugins - geolocation - manual toggle

- call `getWatchID()` - using standard callback, `onSuccess`
 - *get required location details*
 - *then set value for `watchID` property*

```
$("#setWatch").on("change", function(e) {  
    e.preventDefault();  
    //get state of toggle  
    $watchState = $(this).val();  
    //output check of toggle state  
    console.log("watch state is now set to "+$watchState);  
    //check state of toggle  
    if ($watchState === "on") {  
        //call function to start watching...  
        getWatchID();  
        //output check of watchID  
        console.log("watchID = "+watchID);  
    } else {  
        $("#geolocation").empty();  
        //clear the location watching - stops location tracking...  
        navigator.geolocation.clearWatch(watchID);  
        //output check of watchID - check match against on watchID...  
        console.log("clear watch..." + watchID);  
    }  
});
```

- update conditional statement
 - *clear output of coordinates, then clear watching of user's current location*

Image - API Plugin Tester - Geolocation toggle

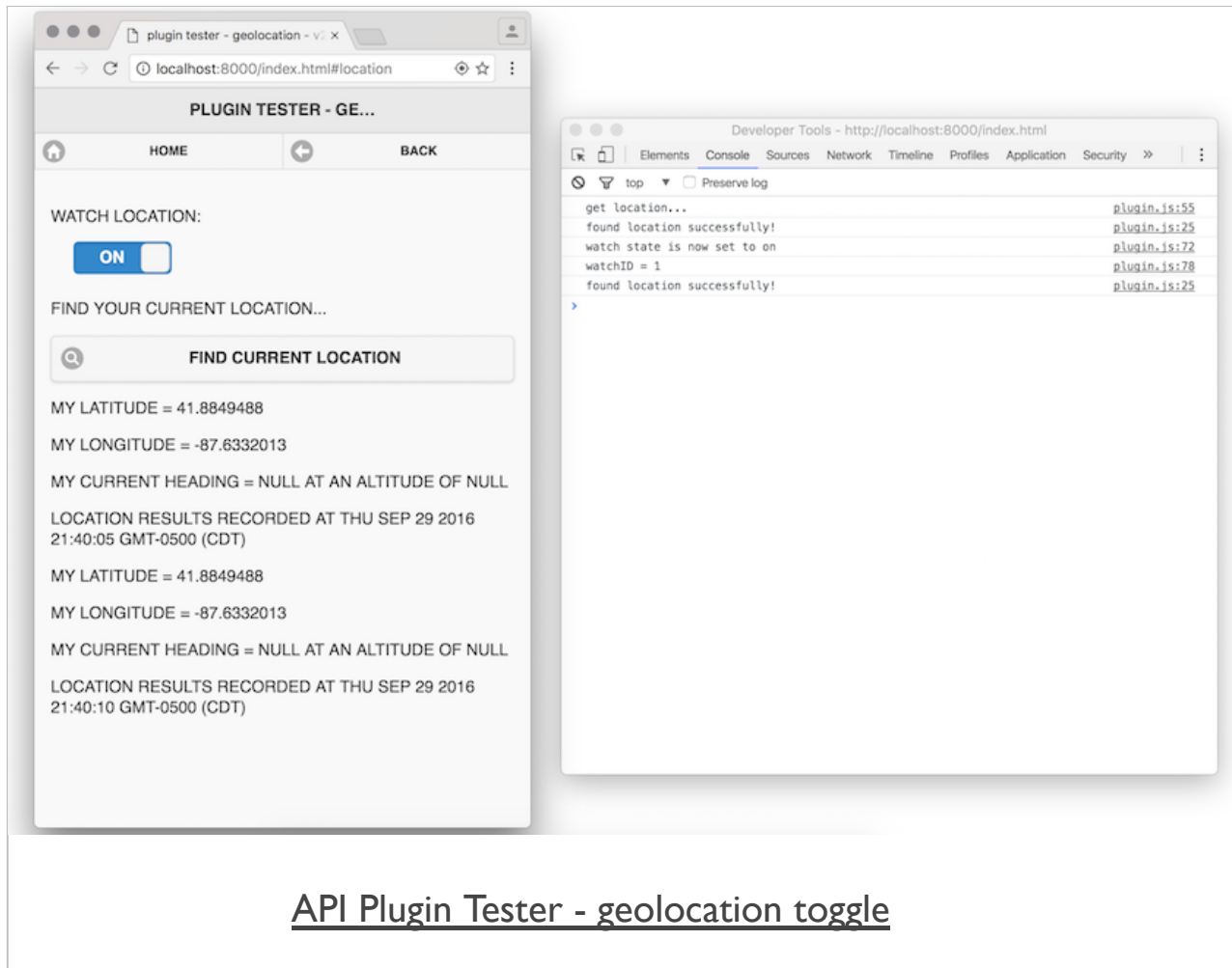
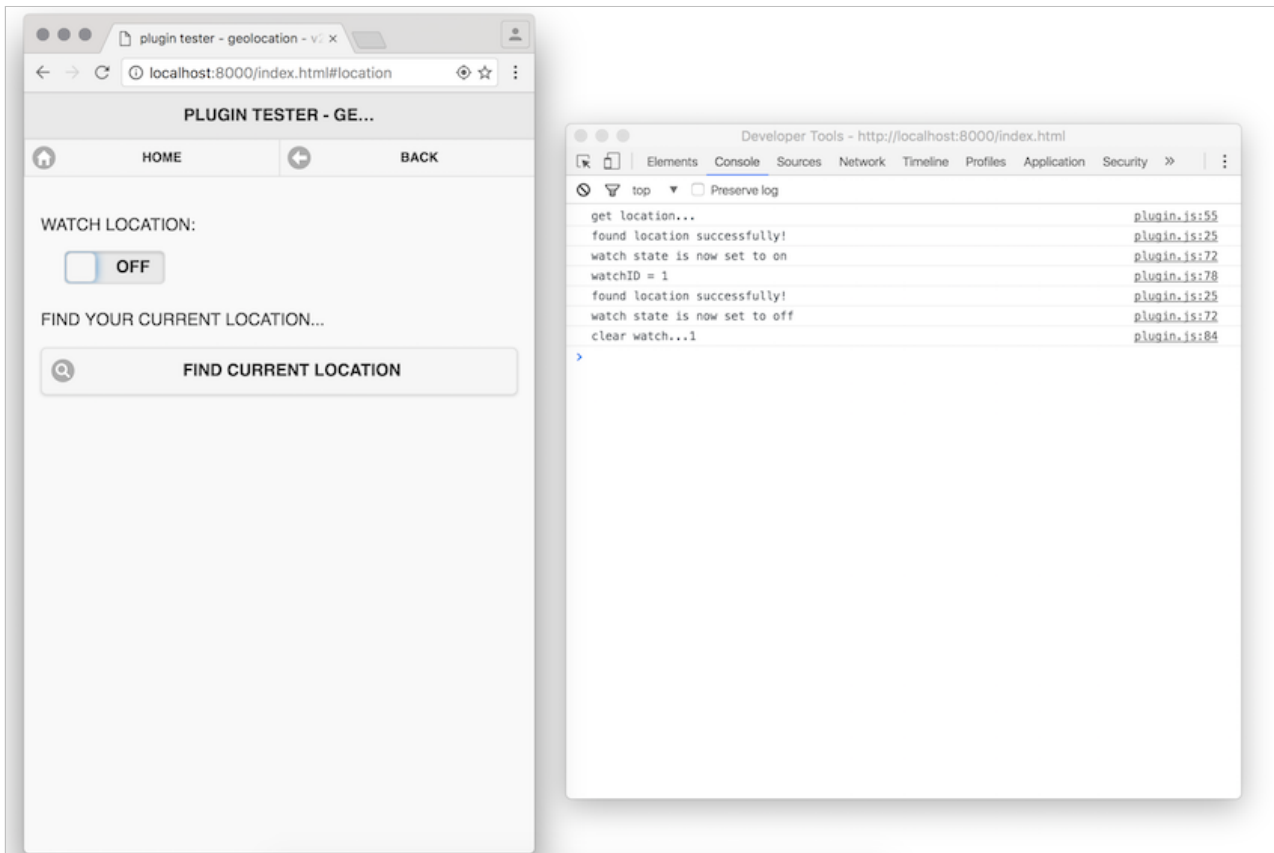


Image - API Plugin Tester - Geolocation toggle



API Plugin Tester - geolocation toggle

Data considerations in mobile apps

- no one size fits all model for mobile
- can't just default to the server-side for reading and writing data
- our app may become useless if we rely heavily on remote data
 - *lose our network connection*
 - *run out of monthly data allowance*
 - *or end up with throttled or restricted data on a poor network, e.g. 2G*
- Facebook's introduction of **2G Tuesdays**
 - *remind employees, developers of 2G limitations and issues around the world*
- also need to consider
 - *data security, read and write privileges for certain data stores, authentication for remote sources...*
- careful consideration of the options for reading and writing data
 - *a crucial aspect of our app's planning and subsequent development*

Cordova app - API plugin examples - plugin test 4

setup

- create our initial plugin test shell application

```
cordova create pluginTest4 com.example.pluginTest4 pluginTest4
```

- add any required platforms, e.g. Android, iOS, Windows Phone...
 - *we'll add iOS as well*

```
cordova platform add android --save
```

- then update the default www directory
- modify the initial settings in our app's `config.xml` file
- then run an initial test to ensure the shell application loads correctly
 - *run in the Android emulator or*
 - *run on a connected Android device*

```
cordova emulate android
```

- or

```
cordova run android
```


Cordova app - API plugin examples - plugin test 4

setup

- also add support for iOS development

```
cordova platform add ios --save
```

- running a test application on iOS is not as simple as Android
- need to add support to Cordova for a local iOS simulator
 - *add package for iOS simulator using **npm***
 - **NB:** *may require admin or sudo permissions to install correctly*

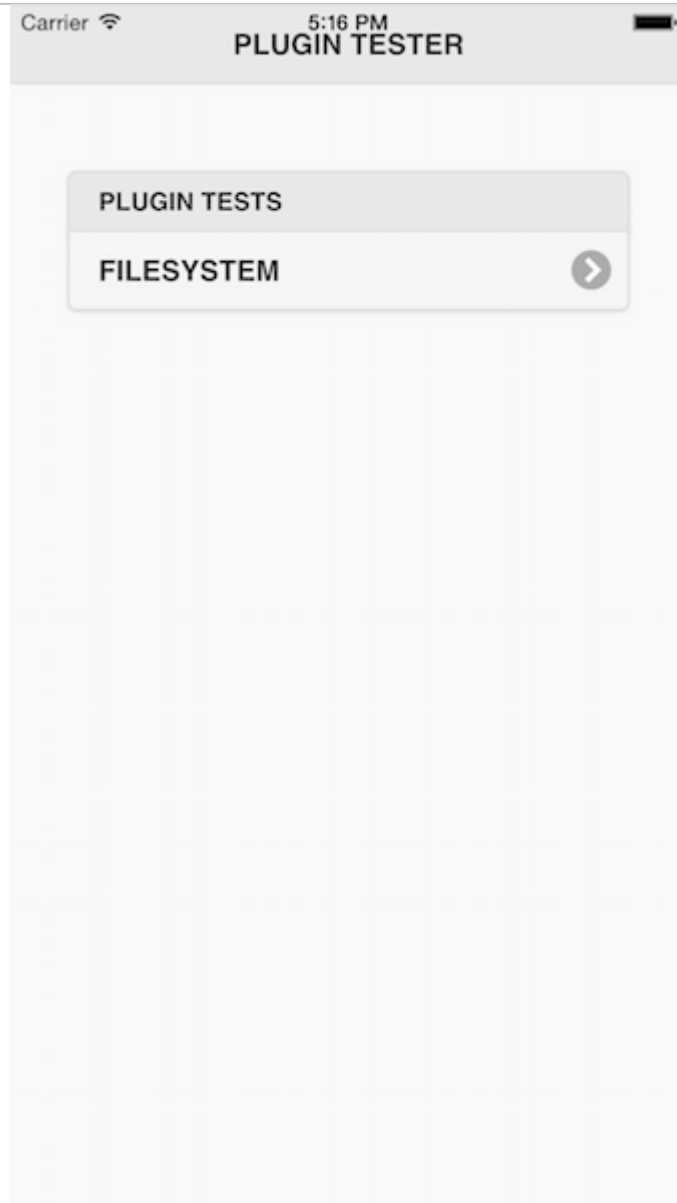
```
npm install -g ios-sim
```

- then run our Cordova app from the working directory

```
cordova run ios
```

- Cordova will try to load the application using this local simulator
 - *without defaulting to Xcode*
- quickly test our iOS application with this simulator

Image - iOS Local Simulator



iOS Simulator - running locally on OS X

Cordova app - API plugin examples - plugin test 4

iOS simulator - options

- iOS simulator gives us many useful options
 - *helpful ways to test our local Cordova based iOS applications*
- emulate many different devices
 - *from the iPhone 6 Plus to the iPad Air*
- mimic many of these device's hardware features
 - *such as rotate, shake, different keyboards...*
 - *also output to a simulated Apple Watch device, 38mm & 42mm*
- various debugging options available within this simulator
 - *including ability to mimic locations for GPS enabled applications*
- quickly take a screenshot of the current application screen within the simulator

Cordova app - API plugin examples - plugin test 4

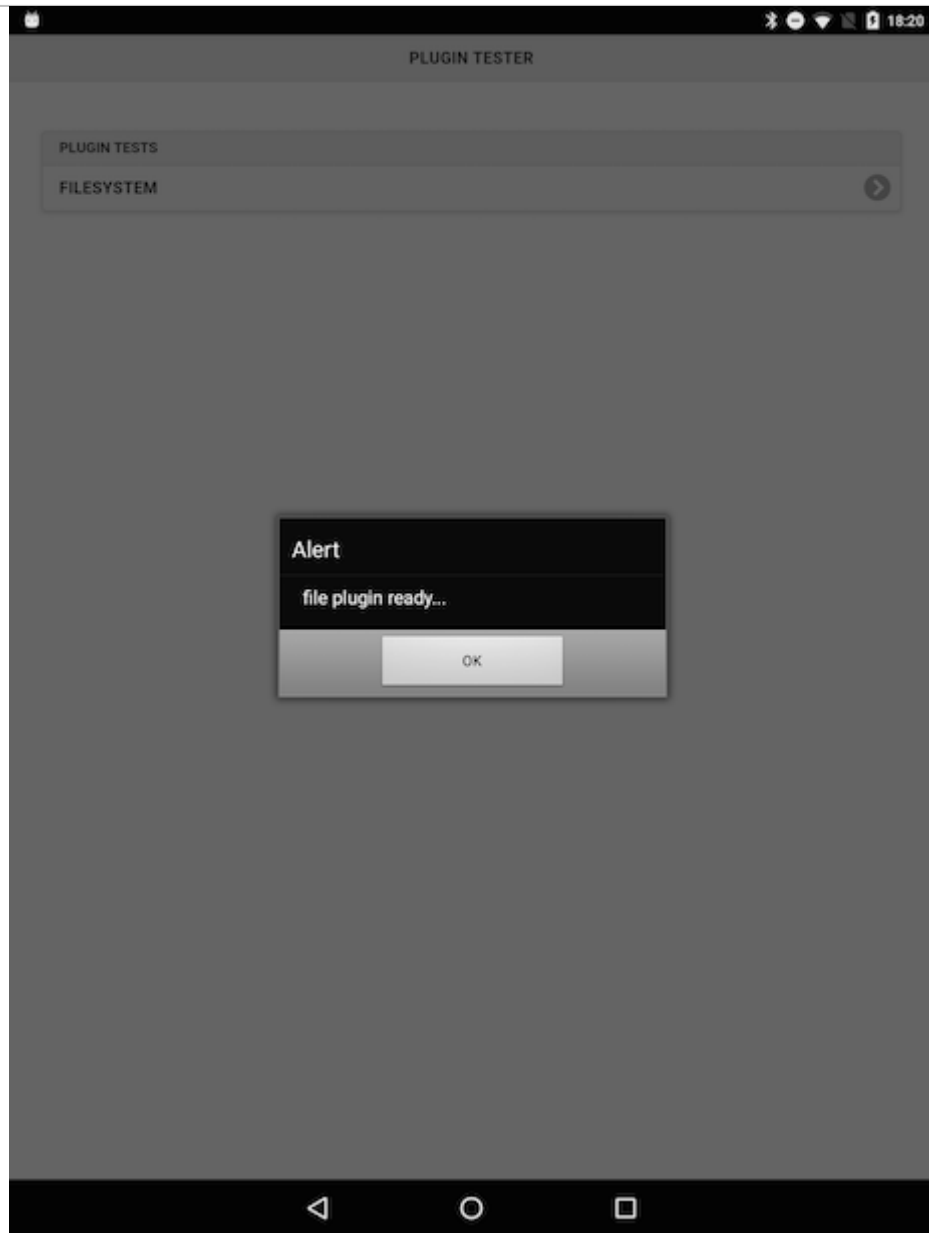
plugins - add filesystem

- add and use the **file** plugin
- plugin has been designed to permit read and write access to files
 - *files are stored on the local device for Cordova applications*
- **file** plugin is initially based on open specifications
 - *includes the **HTML5 File API**, W3's **FileWriter** specification...*
- add the file plugin to our test application using the standard CLI command

```
cordova plugin add cordova-plugin-file
```

- command will install plugin for all currently installed platforms
 - *includes Android and iOS for our test application*

Image - API Plugin Tester - file



API Plugin Tester - file plugin ready

Cordova app - API plugin examples - plugin test 4

plugins - test filesystem

- using this plugin we can read local files from within the filesystem
- we could read a file from within our Cordova application
 - e.g. located in the following directory

```
...  
|- www  
  |- docs  
    |- txt  
      |- madeira.txt
```

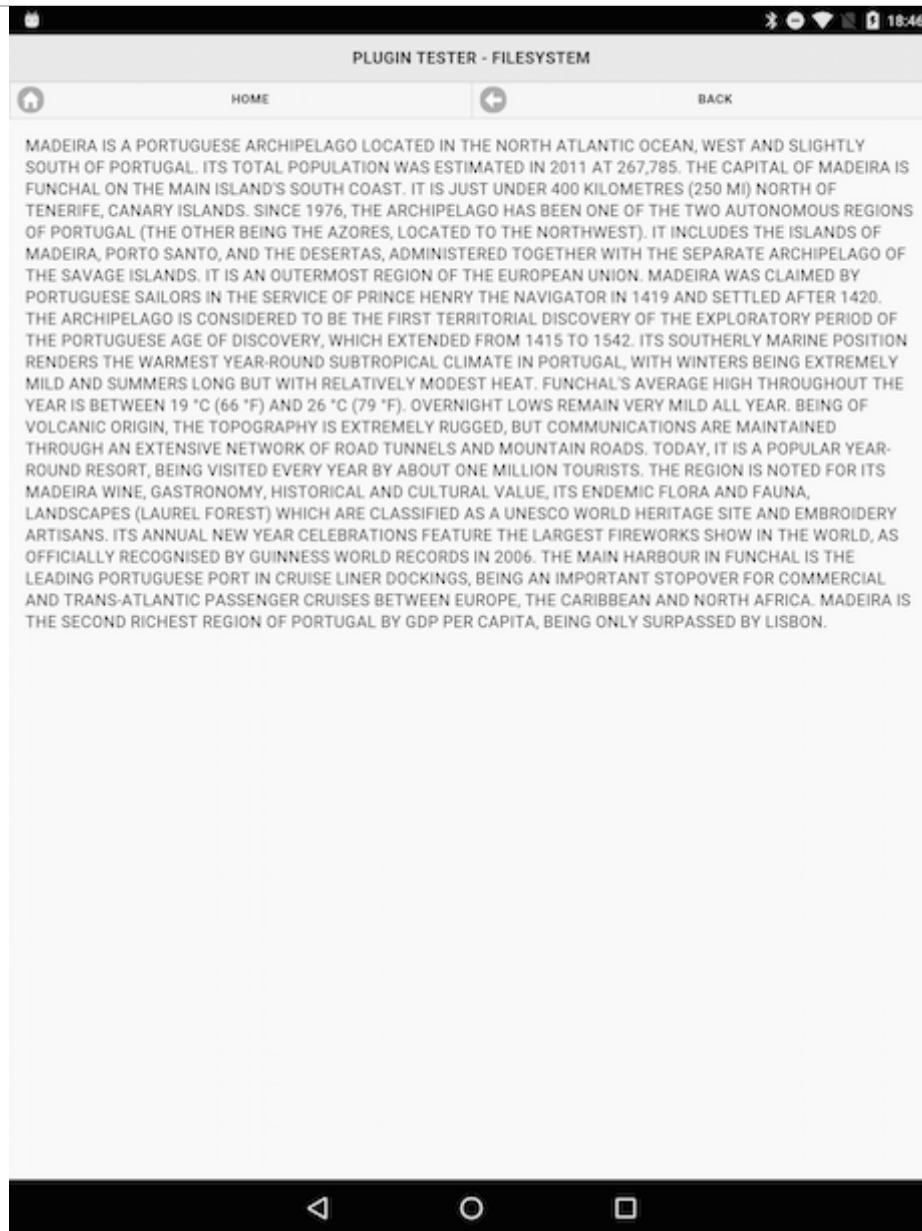
- we can use the available global `cordova.file` object
- to be able to use the URL for our text document in the file-system directory
 - convert it to a *DirectoryEntry* using

```
window.resolveLocalFileSystemURL()
```

- in our standard `onDeviceReady()` function
 - use this global object to resolve the URL of our file
 - then pass to specified callbacks for success and fail

```
window.resolveLocalFileSystemURL(cordova.file.applicationDirectory +  
"www/docs/txt/madeira.txt", onSuccess, onFail);
```

Image - API Plugin Tester - file



API Plugin Tester - read an app txt file

Cordova app - API plugin examples - plugin test 4

plugins - test filesystem onSuccess

- render this text after retrieving from the requested file
 - *update our `onSuccess ()` function to output the file's content*

```
function onSuccess(data) {  
    data.file(function(file) {  
        var readFile = new FileReader();  
        readFile.onloadend = function(e) {  
            //use jQuery selector to add returned file data  
            $("#file-output").html(this.result);  
        }  
        readFile.readAsText(file);  
    });  
}
```

- call the `file ()` method on our returned file data
 - *effectively gives us a hook/handle into the file*
 - *we can now work with the returned file data*
- then call the `FileReader ()` method from the **FileAPI**
 - *and process the returned text*
- output to our specified HTML element
 - *using a standard jQuery selector with the `html ()` method*

Cordova app - API plugin examples - plugin test 4

plugins - test filesystem onFail()

- complement to the `onSuccess()` function
- now add our function `onFail()` for the fail callback
- test it with the returned error code

```
function onFail(error) {  
    console.log("FileSystem Error"+error.code);  
    $("#file-output").html("file plugin error - "+error.code);  
}
```

- uses the passed error object
 - *returns a code for rendering in the specified jQuery selector*
- obviously does not make a lot of sense to our user

Cordova app - API plugin examples - plugin test 4

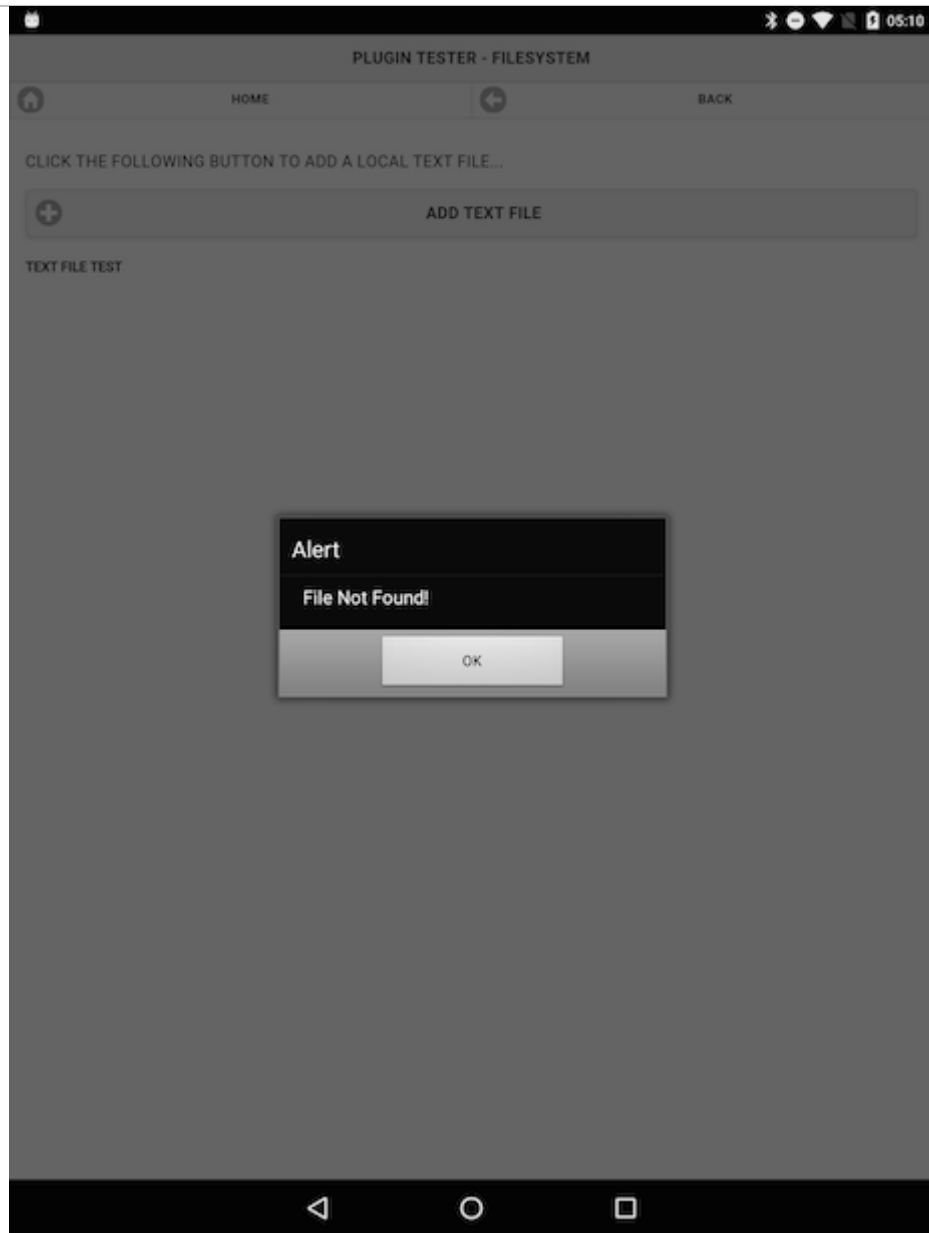
plugins - test filesystem onFail()

- we can use a conditional statement to check for certain returned error codes
 - *then output a meaningful error message to the user in the application*

```
function onFail(error) {  
  switch(error.code) {  
    case 1:  
      alert('File Not Found!');  
      break;  
      //add other options to cover additional error codes...  
    default:  
      alert('An error occurred reading this file.');  };  
}
```

- now output more graceful error messages and feedback to the user
- Web APIs - FileError

Image - API Plugin Tester - file



API Plugin Tester - output error message

Cordova app - API plugin examples - plugin test 4

plugins - test filesystem with event

- easily link file loading to a given event, such as a user tap event
- instead of loading the file by default with the `onDeviceReady()` function
 - *get the contents of our file when needed by the user*
- link this to a button event, a separate page init event...

```
//handle button press for file load  
$("#getFile").on("tap", function(e) {  
    e.preventDefault();  
    getTxtFile();  
});
```

- then call our local file as before within its own function, `getTxtFile()`

Image - API Plugin Tester - file



API Plugin Tester - event file loader

Cordova app - API plugin examples - plugin test 5

plugins - test filesystem with file write

- now read files from the local device's native storage thanks to Cordova's File plugin
- file plugin also offers an option to write to files in the same local filesystem
- quickly create a test app for writing to files
- create your project
- cd to app's working directory
- add required platforms
- add our required Cordova API plugin for working with the file system
- run usual initial tests for app loading, deviceready event...

Cordova app - API plugin examples - plugin test 5

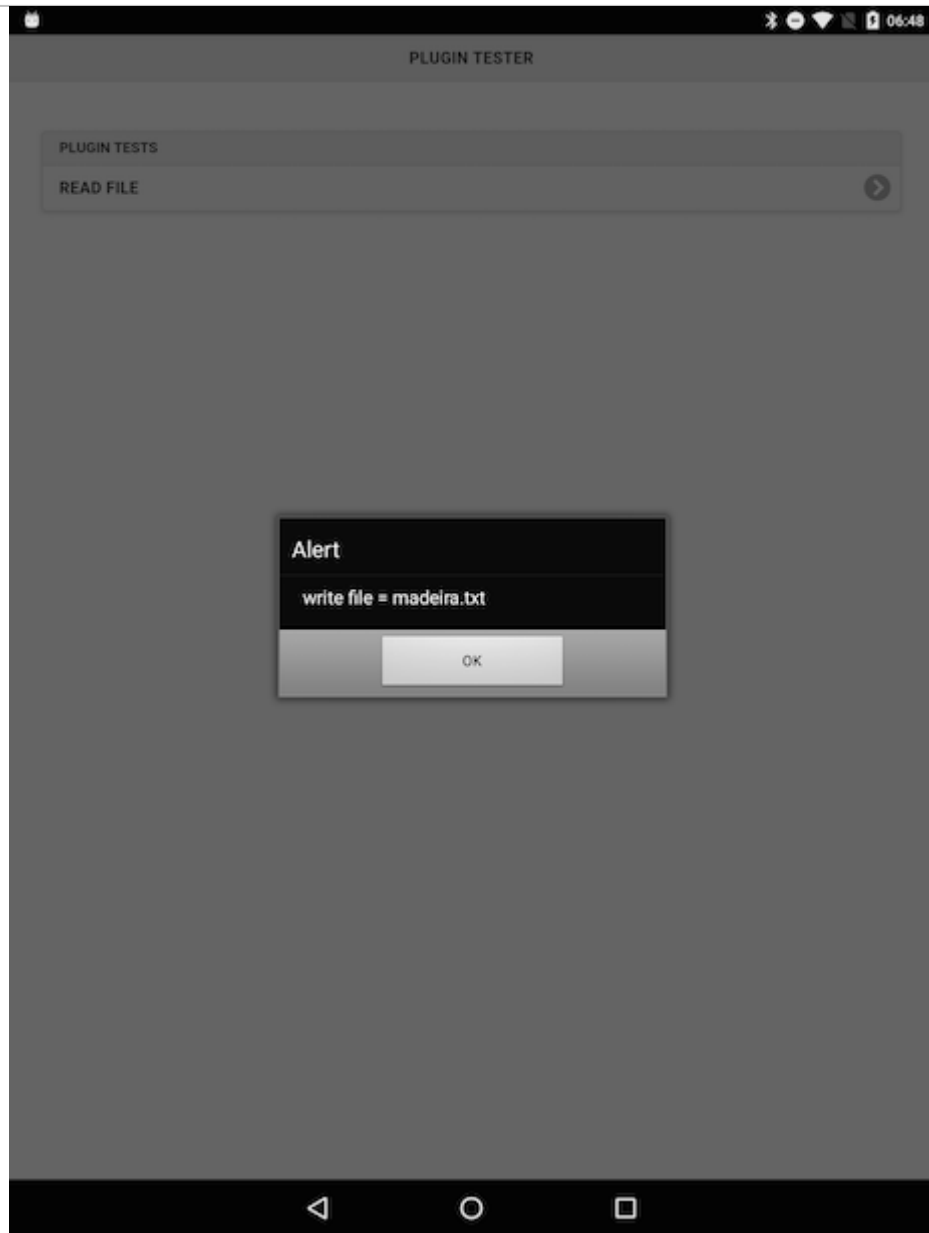
plugins - test filesystem with file write

- now start to add writing to a file to our test app
- start, as we did with file reading, by getting a hook/handle to a file
- we can then write to a file within the assigned app's data directory
 - *specific app directory has read and write access*
 - *allows us to create files as needed for our app*
 - *then read and write within the confines of the native app*
- use `window.resolveLocalFileSystemURL` to allow us to work with this data directory

```
var fileDir = cordova.file.dataDirectory;
window.resolveLocalFileSystemURL(fileDir, function(dir) {
  // do something useful...
});
```

- in application specific directory get our required file for writing

Image - API Plugin Tester - file



API Plugin Tester - get file for writing

Cordova app - API plugin examples - plugin test 5

plugins - test filesystem with file write

- create a new file if it doesn't exist on app loading
- use directory object with `getFile()` method etc...
 - *set flag to create a new file*

```
window.resolveLocalFileSystemURL(fileDir, function(dir) {  
  dir.getFile("madeira.txt", {create:true}, function(file) {  
    //do something useful  
  });  
});
```

- pass file object to other functions for processing...
- create our write function to check and write to specified file within app's data directory

Cordova app - API plugin examples - plugin test 5

plugins - test filesystem with file write

- now write some simple text to our file

```
function writeTxtFile(data) {  
    //check passed data for writing  
    if (data !== "") {  
        //new text to write to file  
        var text = data;  
        //use write file object  
        writeObj.createWriter(function(writeFile) {  
            //call seek() to ensure we append to end of file  
            writeFile.seek(writeFile.length);  
            //create raw Blob for writing  
            var textBlob = new Blob([text], {type: 'text/plain'});  
            //write to the end of the file  
            writeFile.write(textBlob);  
        });  
    }  
}
```

Cordova app - API plugin examples - plugin test 5

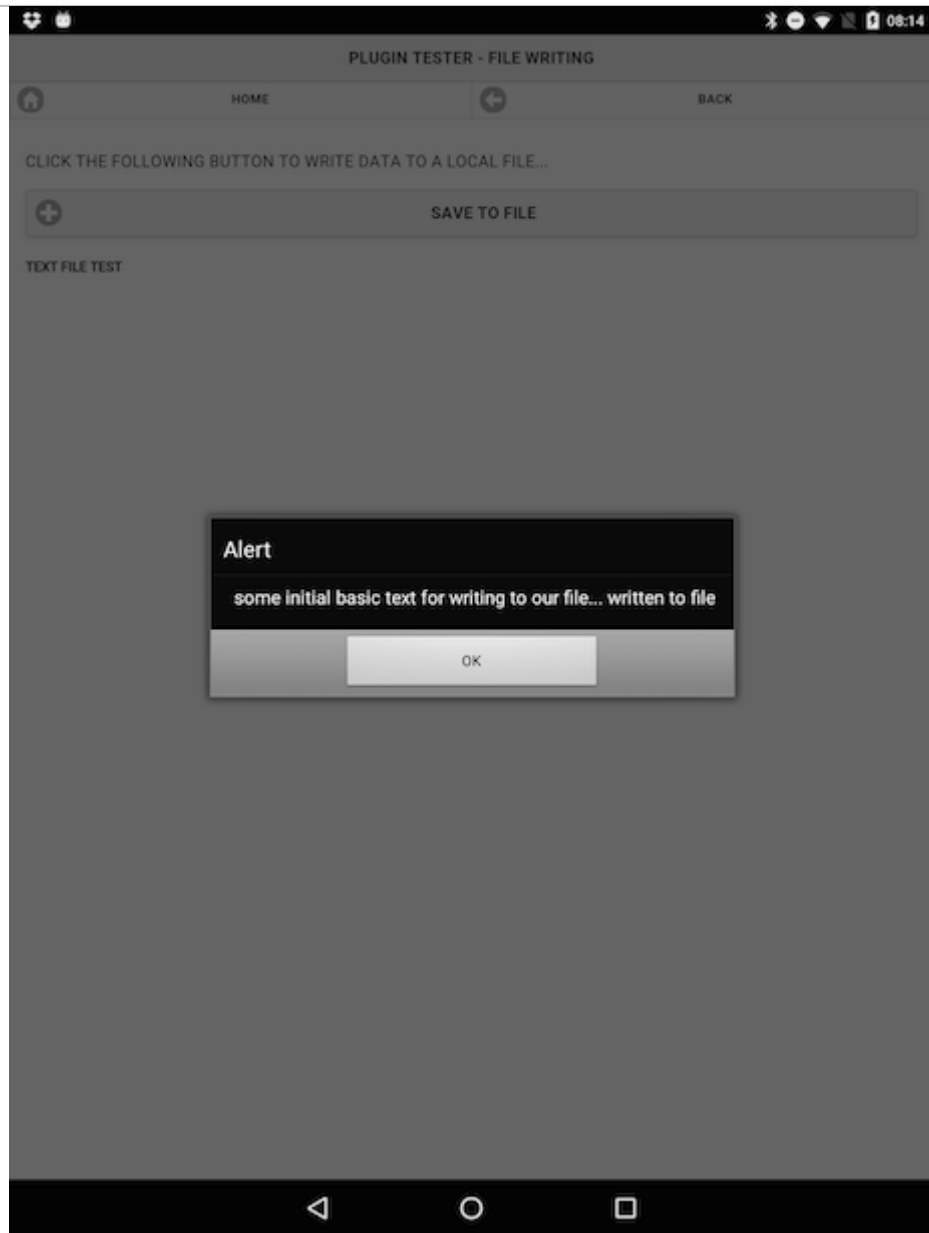
plugins - test filesystem with file write

- then call this `writeTxt()` as needed within our application
 - e.g. calling it from event handler for a button tap

```
//handle button press for file write
$("#saveFile").on("tap", function(e) {
    e.preventDefault();
    writeTxtFile("some initial basic text for writing to our file...");
});
```

- could easily get text to write from an input field, from metadata...
- then pass it to our `writeTxtFile()` function for writing

Image - API Plugin Tester - file



API Plugin Tester - text written to file

Data considerations in mobile apps

- worked our way through Cordova's File plugin
- tested local and remote requests with JSON
- many other options for data storage in mobile applications
- for example

1. LocalStorage

- *based upon the Web Storage API specification*
- *access local data based upon simple key and value pairs*
- *similar concept to Redis*

2. WebSQL

- *offers a full database using tables, queried using SQL*
- *originally rejected by Mozilla and Microsoft's IE team*
- *still widely supported by Chrome and Safari on mobile*
- *MSOpenTech division just released a WebSQL plugin for Cordova*
- *WebSQL support*

3. IndexedDB

- *supposed winner in the WebDB (WebSQL) and Web Simple DB (IndexedDB) wars*
- *still struggles to gain widespread developer support*
- *key/value pairs can often be implemented using LocalStorage*
- *WebSQL, and Sqlite, still popular technologies*

Cordova app - LocalStorage - data test

app setup

- create our initial plugin test shell application

```
cordova create datatest1 com.example.datatest1 DataTest1
```

- add any required plaforms, eg: Android, iOS, Windows...
 - *we'll add iOS as well*

```
cordova platform add android --save
```

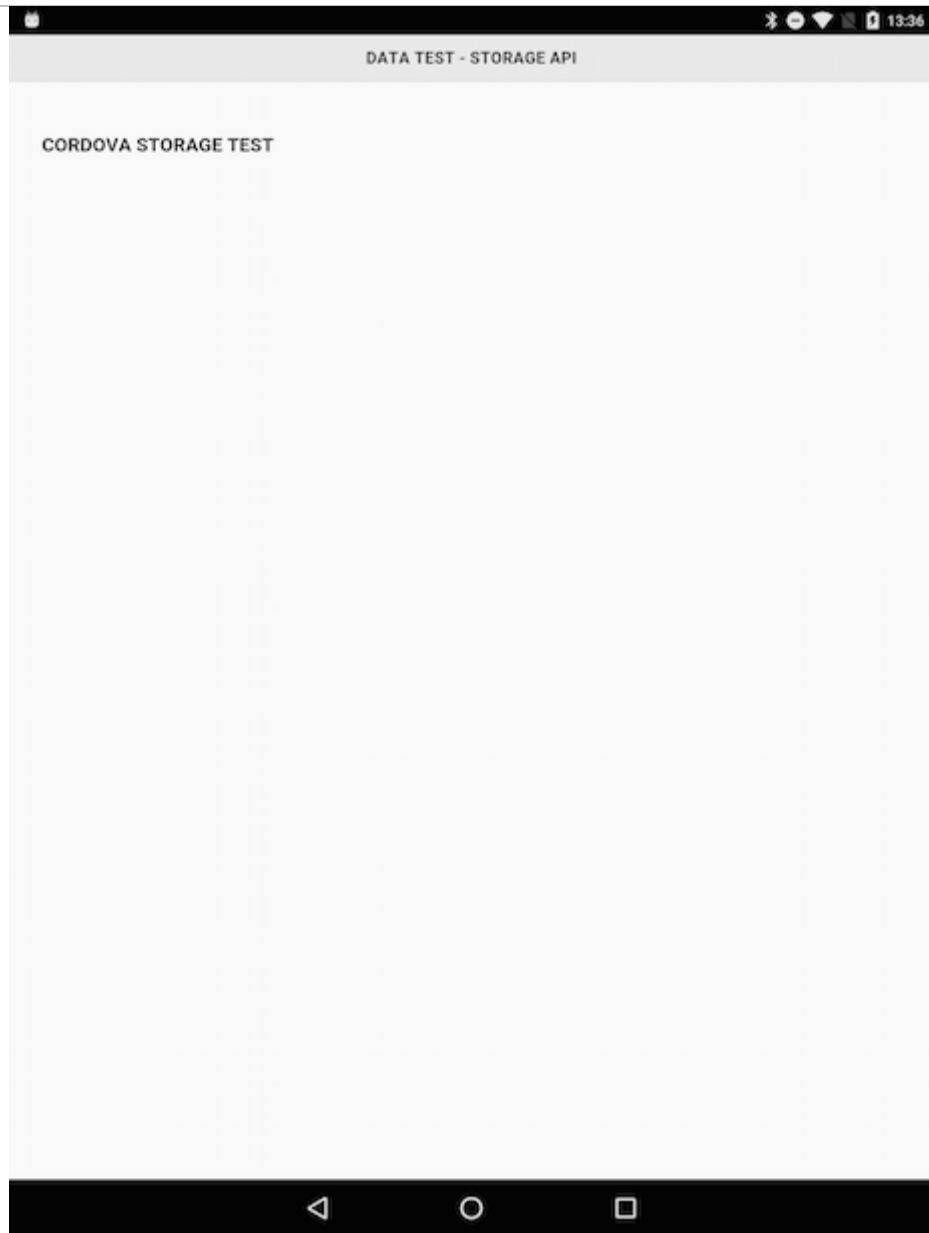
- then update the default www directory
- modify the initial settings in our app's `config.xml` file
- then run an initial test to ensure the shell application loads correctly
 - *run in the Android emulator or*
 - *run on a connected Android device*

```
cordova emulate android
```

- or

```
cordova run android
```

Image - Data Tester



DataTestI - test shell app

Cordova app - LocalStorage - data test

app structure - update HTML

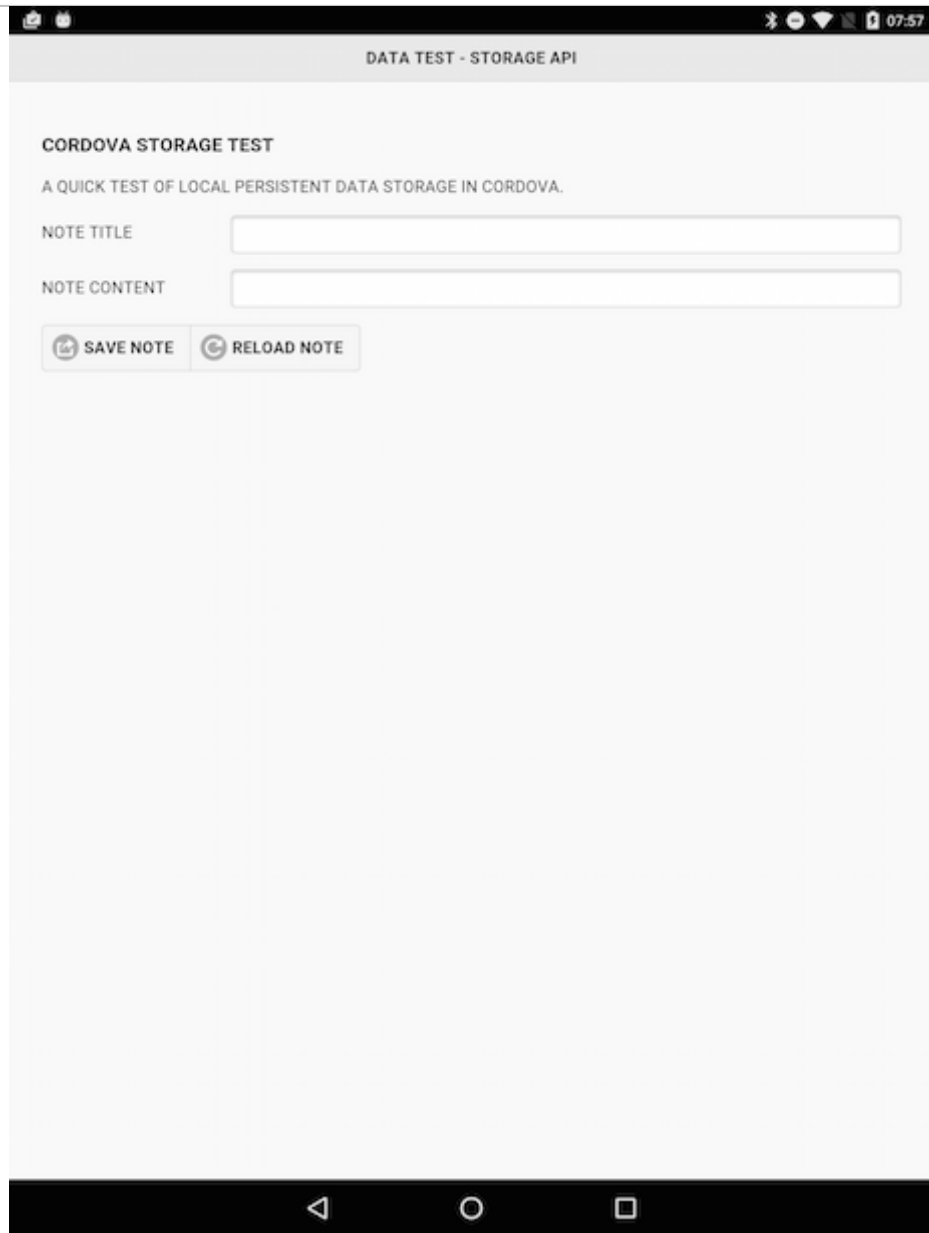
- update app's initial HTML
- home screen includes
 - *basic app headings, app information, and form*
 - *form used for creating, saving, and loading a note*
- HTML form for notes includes
 - *"noteForm" - our form for storing notes*
 - *"noteName" - input text field for title of the note*
 - *"noteContent" - input text field for body of note*
 - *"saveNote" - button to submit data (persist data in storage)*
 - *"reloadNote" - reload the saved note for testing persistence*
 - *"saveResult" - render message from storage request (eg: save successfully)*

Cordova app - LocalStorage - data test

app structure - home screen HTML

```
...
<form id="noteForm">
  <div class="ui-field-contain">
    <label for="noteName">Note Title</label>
    <input type="text" id="noteName" name="noteName"></input>
  </div>
  <div class="ui-field-contain">
    <label for="noteContent">Note Content</label>
    <input type="text" id="noteContent" name="noteContent"></input>
  </div>
  <div data-role="controlgroup" data-type="horizontal">
    <input type="button" id="saveNote" data-icon="action" value="Save Note" data-inline="true"/>
    <input type="button" id="reloadNote" data-icon="refresh" value="Reload Note" data-inline="true"/>
  </div>
</form>
...
```

Image - Data Tester



DATA TEST - STORAGE API

CORDOVA STORAGE TEST

A QUICK TEST OF LOCAL PERSISTENT DATA STORAGE IN CORDOVA.

NOTE TITLE

NOTE CONTENT

DataTestI - setup HTML

Cordova app - LocalStorage - data test

app logic - save.js

- create new JavaScript file to store logic for saving to storage
- name this new JS file, `save.js`
- we can store this in our `/assets/scripts/save.js` directory

```
| - www
  | - assets
    | - scripts
      | - save.js
```

- add our usual `pageinit` event handler
 - *use to register the event handlers for our buttons*
- handlers for `Save Note` and `Reload Note` buttons
- need to validate the form to check for errors...
 - *ensure it meets minimum requirements for saving notes to storage*

Cordova app - LocalStorage - data test

app logic - save.js form validation

- use jQuery's validation plugin to help with form validation
 - *download the plugin's JS file*
 - *add it to our HTML after jQuery file*
- use plugin to define required validation rules for each form field
- use the plugin's `validate()` method to help with this setup
- call the associated `valid()` method to check the passed form

```
$("#noteForm").validate({
  rules: {
    noteName: "required",
    noteContent: "required"
  },
  messages: {
    noteName: "Add title for note",
    noteContent: "Add your note"
  }
});
```

```
if (! $("#noteForm").valid()) {
  return;
}
```

Cordova app - LocalStorage - data test

app logic - save.js

- to save the user created notes
- need to handle the tap event for the Save Note button
- initially check that our form is valid
- validate our form using the `.valid()` method
 - *from the jQuery validation plugin*
- if our form is valid, then the handler can continue
- input text values for both `noteName` and `noteContent`
 - *now set as attributes in a JSON object*
 - *convert this object to a string using `JSON.stringify()`*
- persist this stringified JSON object in the device's local storage
- use the app's main object
 - *set a key and a value pair for notes in persistent storage*

Cordova app - LocalStorage - data test

app logic - save.js - save button handler

- event handler for save button

```
// handler for save note button
$("#saveNote").on("tap", function(e) {
    e.preventDefault();
    //check form is valid
    if (! $("#noteForm").valid()) {
        return;
    }
    //store notes
    storageNotes.set(NOTE_KEY, JSON.stringify({
        noteName: $("#noteName").val(),
        noteContent: $("#noteContent").val()
    }));
});
// inform user note saved
$("#saveResult").html("note saved...");
});
```

- main app object

```
var storageNotes = NotesManager.getInstance();
```

Image - Data Tester

DATA TEST - STORAGE API

CORDOVA STORAGE TEST

A QUICK TEST OF LOCAL PERSISTENT DATA STORAGE IN CORDOVA.

NOTE TITLE

NOTE CONTENT

NOTE SAVED...

DataTestI - save a note

Cordova app - LocalStorage - data test

app logic - save.js

- need to handle events for our `reloadNote` button
- retrieve our notes data
 - loaded by calling the *`reloadNoteData()`* function
- uses the main app object, `storageNotes`
 - gets the defined key for our notes
- use this key to retrieve stored *stringified* JSON object
- then use `JSON.parse()` to convert the *stringified* object to a plain JSON object
 - contains our note information
- use this note information
 - populate form fields
 - output our notes for rendering to the DOM

Cordova app - LocalStorage - data test

app logic - save.js - reload button handler

- event handler for reload button

```
// handler for reload note button
$("#reloadNote").on("tap", function(e) {
    e.preventDefault();
    reloadNoteData();
    $("#saveResult").html("note reloaded...");
});
```

- reload note data

```
function reloadNoteData() {
    var noteInfo = JSON.parse(storageNotes.get(NOTE_KEY));
    loadFormFields(noteInfo);
    noteOutput(noteInfo);
}
```

- load form fields data

```
function loadFormFields(data) {
    if (data) {
        $("#noteName").val(data.noteName);
        $("#noteContent").val(data.noteContent);
    }
}
```

Cordova app - LocalStorage - data test

app logic - save.js

- pageinit event
 - *eg: check and validate the rendered form for our notes*
- to validate our form we specify
 - *a set of options as a parameter to `validate()`*
 - *many different options available*
 - *eg: add a `rules` object, `messages` object...*
- in the `rules` object
 - *set both input fields as required*
- then reload our note data
 - *update the application accordingly*

Cordova app - LocalStorage - data test

app logic - save.js - pageshow event

```
$("#noteForm").validate({
  rules: {
    noteName: "required",
    noteContent: "required"
  },
  messages: {
    noteName: "Add title for note",
    noteContent: "Add your note"
  }
});
```

Cordova app - LocalStorage - data test

app logic - storagenotes.js

- add another new JS file, `storagenotes.js`
 - *store the logic for getting and setting of data with `localStorage`*
- start by creating a singleton object for this instance
- creating this object to ensure that we only have one instance
- create this object by calling the `getInstance()` function
 - *in effect, the guardian to the instance object for the application*
- function also highlights a pattern known as `Lazy Load`
 - *checks to see if an instance has already been created*
- if not, create one and then store for future reference
- all subsequent calls will now received this stored reference
- this pattern is particularly useful for mobile development
- helps us save CPU and memory usage within an application
 - *an object is only created when it is actually needed*
- gives us a single object with getters and setters for the local storage

Cordova app - LocalStorage - data test

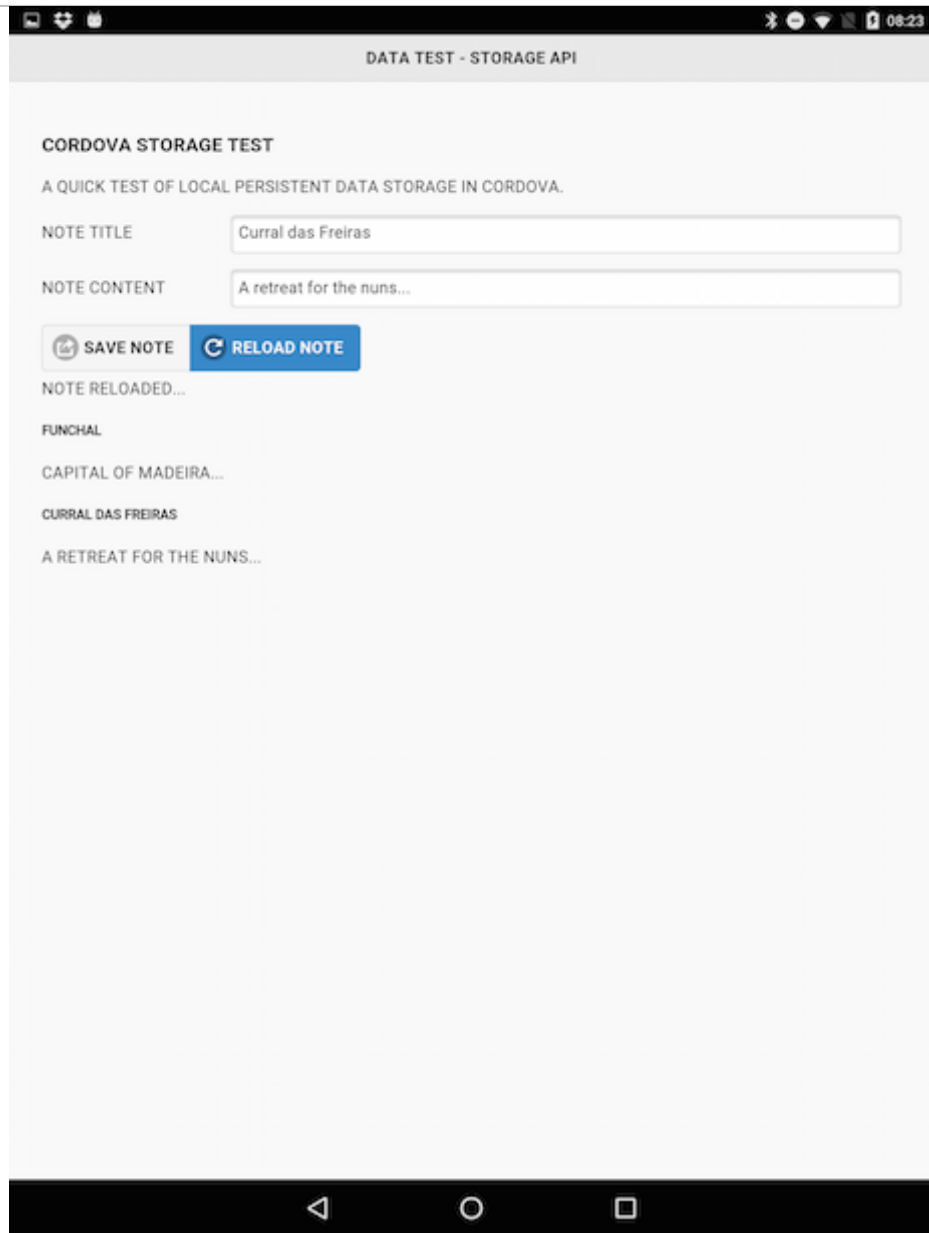
app logic - storagenotes.js

```
var NotesManager = (function () {
    var instance;

    function createNoteObject() {
        return {
            set: function (key, value) {
                window.localStorage.setItem(key, value);
            },
            get: function (key) {
                return window.localStorage.getItem(key);
            }
        };
    };

    return {
        getInstance: function () {
            if (!instance) {
                instance = createNoteObject();
            }
            return instance;
        }
    };
})();
```

Image - Data Tester



DataTestI - update the notes

References

- Cordova API
 - *plugin - filesystem plugin*
 - *plugin - file transfer plugin*
 - *plugin - geolocation*
 - *Storage*
- GitHub
 - *cordova-plugin-file*
- HTML5
 - *HTML5 File API*
- MDN
 - *Web APIs - FileError*
- W3C
 - *Geolocation API Specification*