Comp 322/422 - Software Development for Wireless and Mobile Devices

Fall Semester 2017 - Week 4

Dr Nick Hayward

Cordova app - API plugin examples

- a few API plugins to consider
  - *accelerometer*
  - *camera*
  - *connection*
  - *device*
  - *file*
  - *geolocation*
  - *InAppBrowser*
  - *media and capture*
  - *notification*
  - *StatusBar*
  - ...

Cordova app - API plugin examples - plugin test 2

setup

- create our initial plugin test shell application

```
cordova create plugintest2 com.example.plugintest plugintest2
```

- add any required plaforms, eg: Android, iOS, Windows...

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

- 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
```

- then start to update the default `www` directory
- modify the initial settings in our app's `config.xml` file

Cordova app - API plugin examples - plugin test 2

application structure

- might update our initial Cordova template
  - *better structure for plugin test application*
  - *structure might look as follows*

```
|- hooks
|- platforms
   |- android
   |- platforms.json
|- plugins
   |- cordova-plugin-whitelist
   |- android.json
   |- fetch.json
|- res
   |- icon
   |- splash
|- www
   |- assets
      |- images
      |- scripts
      |- styles
   |- docs
      |- json
      |- txt
      |- xml
   |- media
      |- audio
      |- images
      |- video
   |- index.html
|- config.xml
```

Cordova app - templates - basic

- Cordova default template for project structure
  - *`create` command used for basic structure...*
- create custom, reusable template for a new project
  - *e.g. create starting template for tabs, menu &c. based app...*
- to create a custom template
  - *start with new project structure for Cordova*
  - *then modify to create and configure app structure*
  - *set required icons, splashscreens, designs &c. for template*
- then we can start to package a reusable template

Cordova app - templates - structure

- each template uses the following directory structure

```
|-- template_package
    |__ package.json
    |__ index.js
    |__ template_src
        |__ ... (app template contents...)
```

- template specific code is added to `template_src` directory
- `package.json` includes reference to template's `index.js` file
- `index.js` used to export reference to `template_src` directory

Cordova app - templates - `template_src`

- `template_src` usually includes the following structure

```
|-- hooks (add custom hooks for template, app &c...)
|-- www
    |__ css
        |__ index.css
    |__ img
        |__ logo.png
    |__ js
        |__ index.js
    |__ index.html
|-- config.xml
```

- add any custom scripts to the `hooks` directory
- design and build our template in the `www` directory
- `template_src/config.xml` will usually follow pattern of default Cordova config
- then add template customisations, e.g.
  - *name, description, icons, splashscreens...if necessary*

Cordova app - templates - `package.json`

- `package.json` includes template specific metadata
  - *add keyword `cordova:template` & `ecosystem:cordova`*
  - *used for package distribution, e.g. NPM*
- add reference to `index.js`

```
"main": "index.js"
```

- output will be similar to a standard NPM `package.json` file
  - *created for NPM package management*
  - *then initialised using the command,*

```
npm init
```

Cordova app - templates - template `index.js`

- then add necessary export reference for `template_src` to our template `index.js` file
  - *follows a standard pattern*

```
var path = require('path');

module.exports = {
    dirname : path.join(__dirname, 'template_src')
};
```

Cordova app - templates - finish & create

- template is now ready to be published and shared online
  - *use NPM, GitHub, &c.*
- use as the template for a new local project

```
cordova create basic com.example.basic BasicTemplate --template <path-to-template>
```

- add the local directory path for the custom template
  - *replace `<path-to-template` with local directory for template...*
- creates new Cordova project with custom template
  - *uses `template_src` for the project*

Cordova app - API plugin examples - plugin test 2

plugins - add camera plugin

- now add the camera plugin to our test application
- two ways we can add camera functionality to our application
  - *use the camera plugin*
  - *use the more generic Media Capture API*
- main differences include
  - ***camera*** *plugin focuses on camera capture and functionality*
  - ***media capture*** *includes additional options such as video and audio recording*
- add the camera plugin using the following Cordova CLI command

```
cordova plugin add cordova-plugin-camera
```

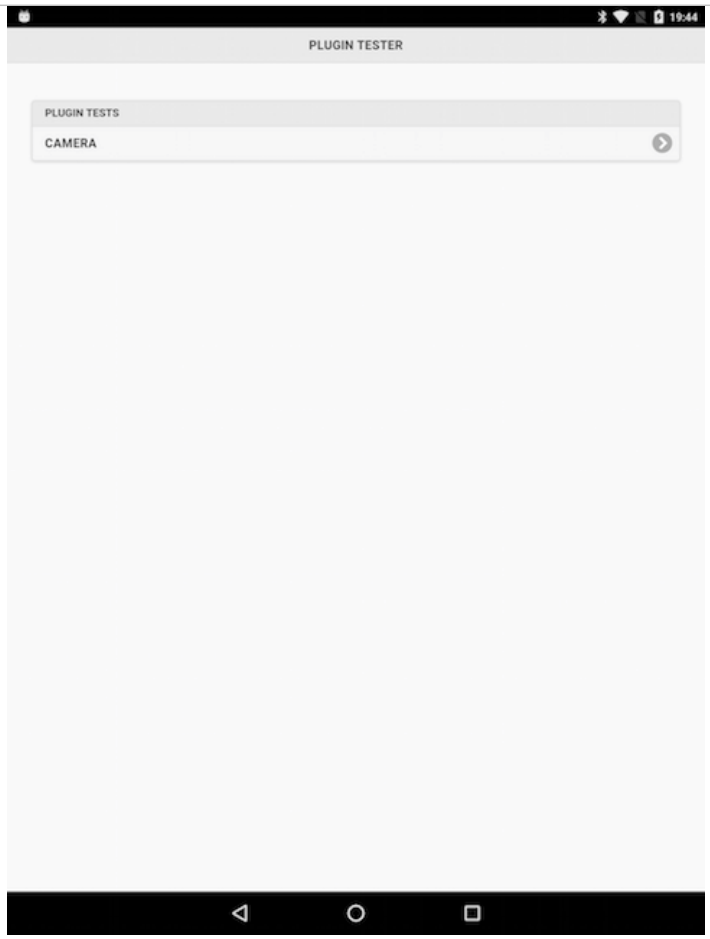
- provides standard `navigator` object
  - *enables taking pictures, and choose images from local image library*

Cordova app - API plugin examples - plugin test 2

plugins - add camera page

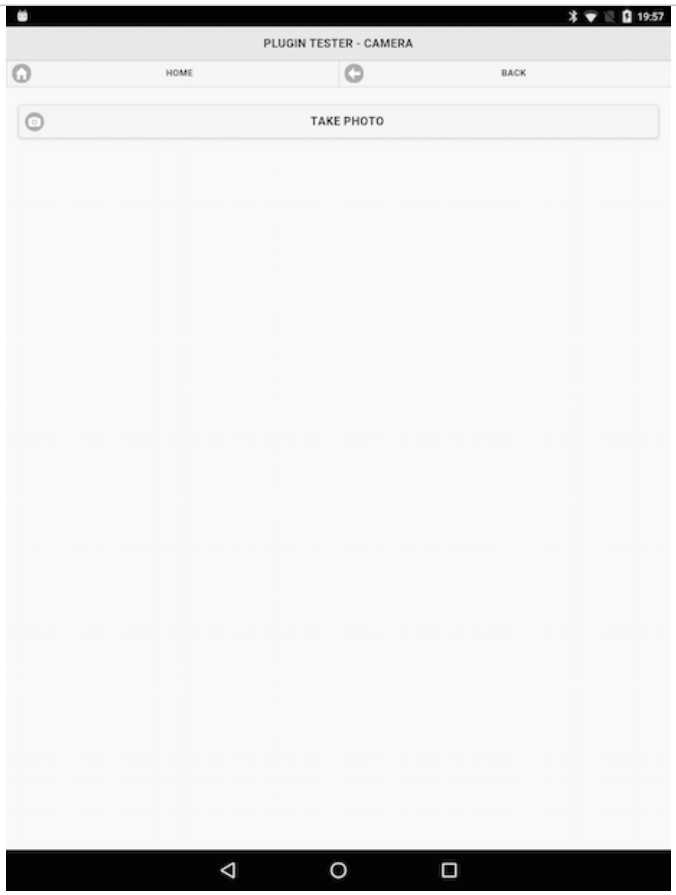
```
<!-- camera page -->
<div data-role="page" id="camera">
  <div data-role="header">
    <h3>plugin tester - camera</h3>
    </div><!-- /header -->
  <div data-role="navbar" data-iconpos="left">
    <ul>
      <li><a class="ui-btn" data-icon="home" data-transition="slide" href="#home">home</a></li>
      <li><a class="ui-btn" data-icon="arrow-l" data-rel="back">back</a></li>
    </ul>
  </div><!-- /navbar -->
  <div data-role="content">
    <input type="button" id="takePhoto" data-icon="camera" value="Take Photo" />
    <div id="photo">
      <img id="imageView" style="width: 100%;"></img>
    </div><!-- /photo -->
    <div data-role="popup" id="photoSelector" style="min-width: 250px;">
      <ul data-role="listview" data-inset="true">
        <li data-role="divider">Choose Photo</li>
        <li><a id="cameraPhoto" href="#">Take Photo with Camera</a></li>
        <li><a id="galleryPhoto" href="#">Get Photo from Gallery</a></li>
      </ul>
    </div><!-- /photoSelector -->
  </div><!-- /content -->
</div><!-- /camera page -->
```

Image - API Plugin Tester - Home



API Plugin Tester - initial home page

Image - API Plugin Tester - Camera



API Plugin Tester - initial camera page

Cordova app - API plugin examples - plugin test 2

plugins - add camera logic

- basic UI is now in place
- start to add some logic for taking photos with the device's camera
- need to be able to get photos from the device's image gallery
- app's logic in initial `plugin.js` file
- handlers for the tap events
  - *a user tapping on the **takePhoto** button*
  - *then the options in the **photoSelector***
  - *take a photo with the camera*
  - *get an existing photo from the gallery*
- use the `onDeviceReady()` function
  - *add our handlers and processors for both requirements*
  - *add functionality for camera and gallery components*

Cordova app - API plugin examples - plugin test 2

plugins - add camera logic

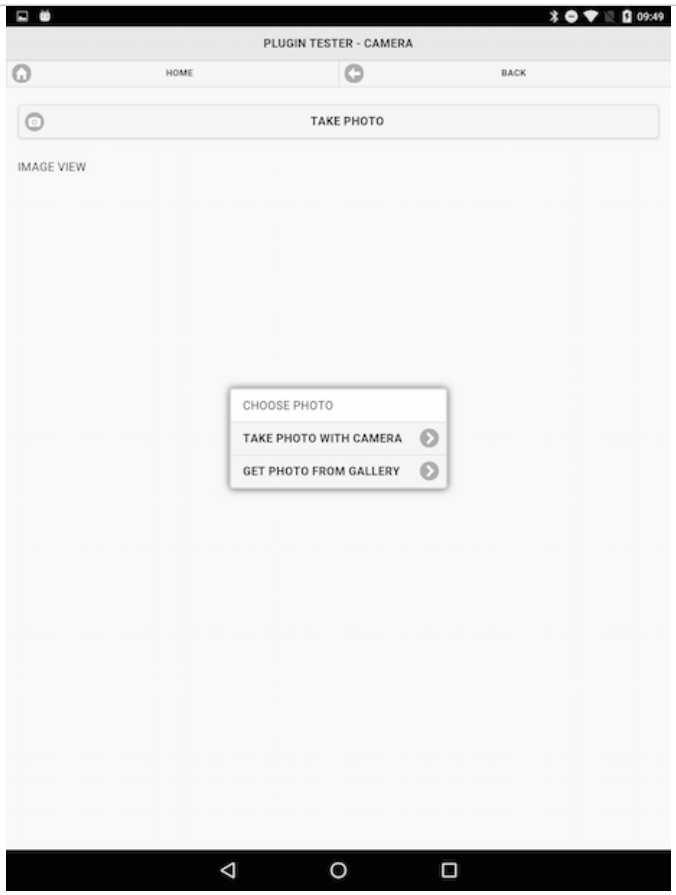
- add our handlers for the tap events
- initial handlers for `takePhoto`, `cameraPhoto`, and `galleryPhoto`

```
$("#takePhoto").on("tap", function(e) {
  e.preventDefault();
  //show popup options for camera
  $("#photoSelector").popup("open");
})

$("#cameraPhoto").on("tap", function(e) {
  e.preventDefault();
  //hide popup options for camera
  $("#photoSelector").popup("close");
})

$("#galleryPhoto").on("tap", function(e) {
  e.preventDefault();
  //hide popup options for camera
  $("#photoSelector").popup("close");
})
```

Image - API Plugin Tester - Camera



API Plugin Tester - camera photo selector

Cordova app - API plugin examples - plugin test 2

plugins - add camera logic

- capture an image using this plugin with the native device's camera hardware
- use the provided `navigator` object for the camera
  - *then call the `getPicture` function*
- also specify required callback functions for the camera
  - *and add some required options for quality...*

```
//Use from Camera
navigator.camera.getPicture(onSuccess, onFail, {
  quality: 50,
  sourceType: Camera.PictureSourceType.CAMERA,
  destinationType: Camera.DestinationType.FILE_URI
});
```

- quality option has been reduced to `50` for testing
  - *choose a value between 0 and 100 for our final application*
  - *100 being original image file from the camera*
- option for `destinationType` now defaults to `FILE_URI` could be changed to `DATA_URL`
  - ***NB:*** *`DATA_URL` option can crash an app due to low memory, system resources...*
  - *returns a base-64 encoded image*
  - *then render in a chosen format such as a JPEG*

Cordova app - API plugin examples - plugin test 2

plugins - add camera logic

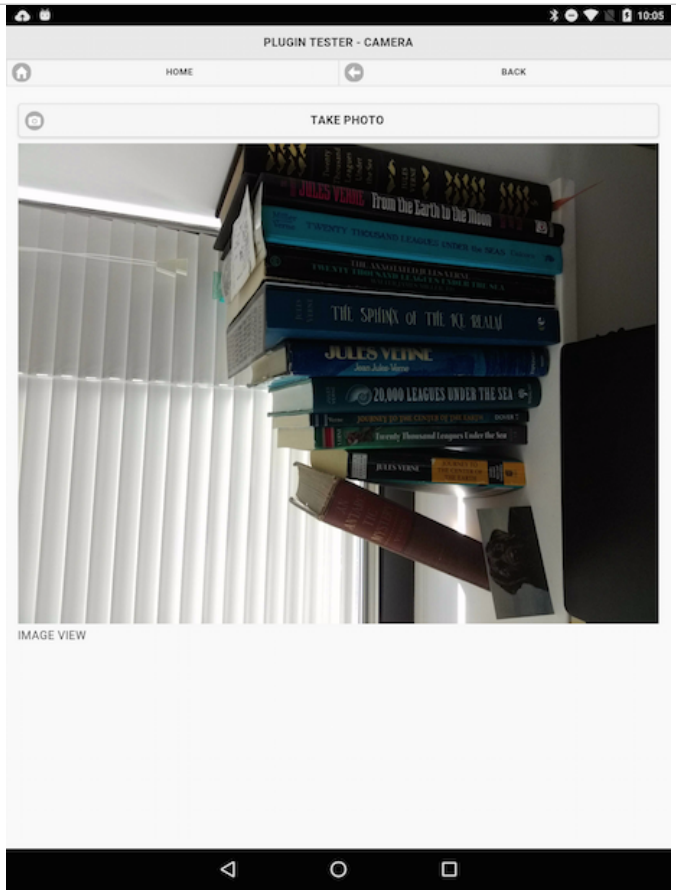
- two callback functions are `onSuccess` and `onFail`
  - *set logic for returned camera image and any error message*

```
function onSuccess(imageData) {
  //JS selector faster than jQuery...
  var image = document.getElementById('imageView');
  image.src = imageData;
}

function onFail(message) {
  alert('Failed because: ' + message);
}
```

- `onSuccess` function accepts a parameter for the returned image data
- using returned image data to output and render our image in the test `imageView`
- `onFail` function simply outputting a returned error message
- we can use these two callback functions to perform many different tasks
  - *we can pass the returned image data to a save function, or edit option...*
  - *they act like a bridge between our own logic and the native device's camera*

Image - API Plugin Tester - Camera



API Plugin Tester - image rotated

Cordova app - API plugin examples - plugin test 2

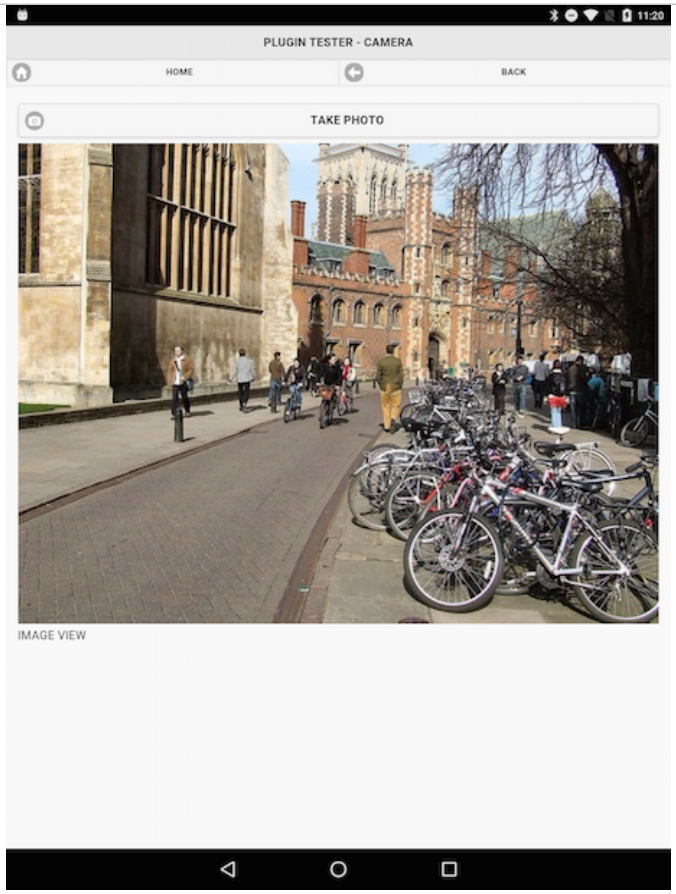
plugins - update camera logic

- returned an image from the camera
- update our application to select an image from gallery application
- add a conditional check to our `getPhoto()` function
  - *allows us to differentiate between a camera or gallery request*

```
navigator.camera.getPicture(onSuccess, onFail, {
  sourceType: Camera.PictureSourceType.PHOTOLIBRARY,
  destinationType: Camera.DestinationType.FILE_URI
});
```

- update in the `sourceType` from `CAMERA` to `PHOTOLIBRARY`
- returned image respects original orientation of gallery image

Image - API Plugin Tester - Camera



API Plugin Tester - image from gallery

Cordova app - API plugin examples - plugin test 2

plugins - fix camera logic

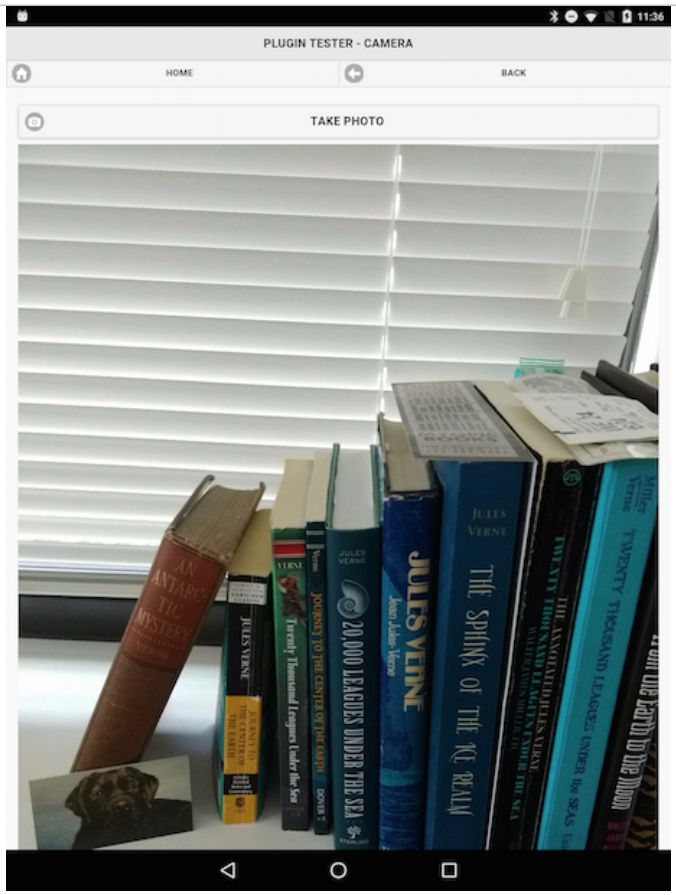
- need to fix the orientation issue with the returned image from the camera
- options for this plugin make it simple to update our logic for this requirement
  - *add a new option for the camera*

```
correctOrientation: true
```

- ensures that the original orientation of the camera is enforced
- updated logic is as follows

```
//Use from Camera
navigator.camera.getPicture(onSuccess, onFail, {
  quality: 50,
  correctOrientation: true,
  sourceType: Camera.PictureSourceType.CAMERA,
  destinationType: Camera.DestinationType.FILE_URI
});
```

Image - API Plugin Tester - Camera



API Plugin Tester - correct image orientation

Cordova app - API plugin examples - plugin test 2

plugins - camera updates

- continue to add many other useful options
  - *specifying front or back cameras on a device*
  - *type of media to allow*
  - *scaling of returned images*
  - *edit options...*
- in the app logic, also need to abstract the code further
  - *too much repetition in calls to the `navigator` object for the camera*
- then add more options and features
  - *save, delete, edit options*
  - *organise our images into albums*
  - *add some metadata for titles etc*
  - *add location tags for coordinates...*

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 3

setup

- create our initial plugin test shell application

```
cordova create plugintest3 com.example.plugintest plugintest3
```

- add any required plaforms, 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 3

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*

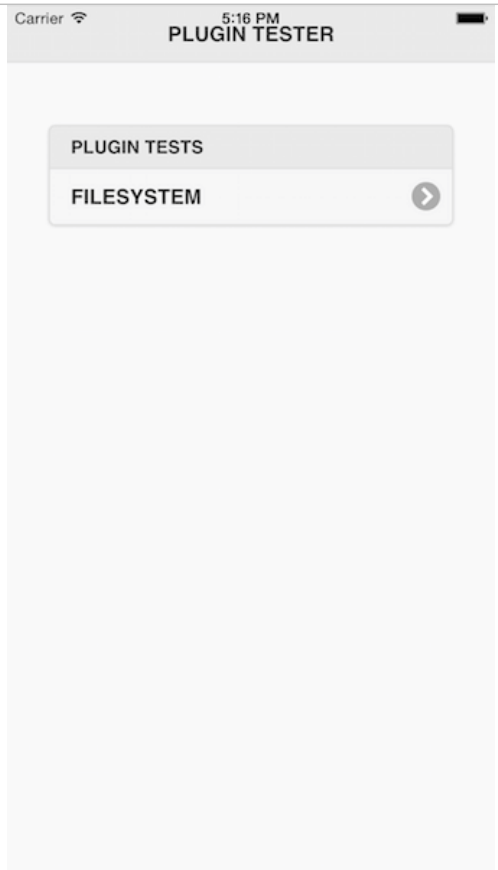
```
npm install -g ios-sim
```

- 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 3

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 3

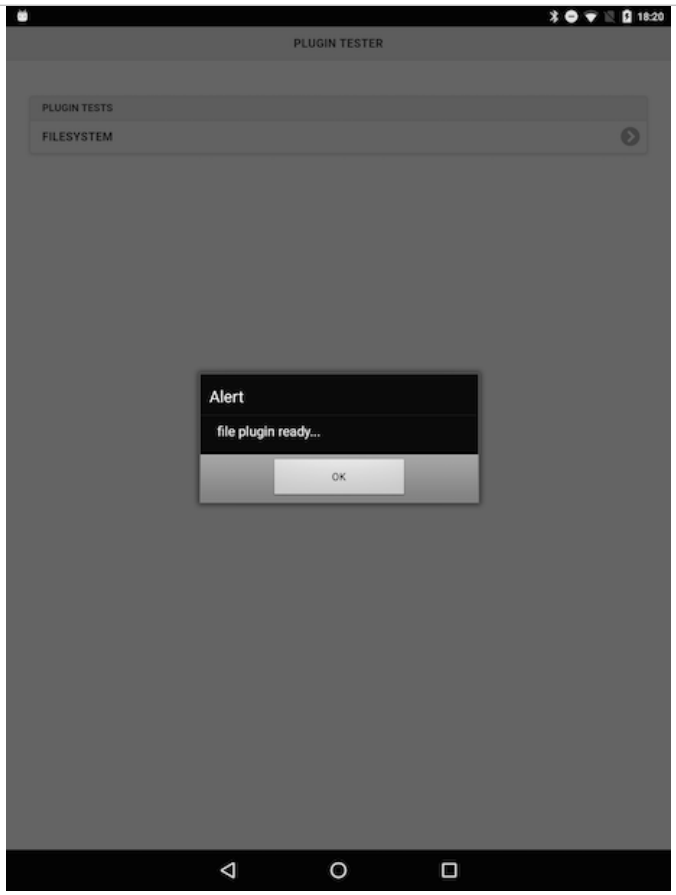
plugins - add filesystem

- 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**, W3C'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 3

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 3

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 3

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 3

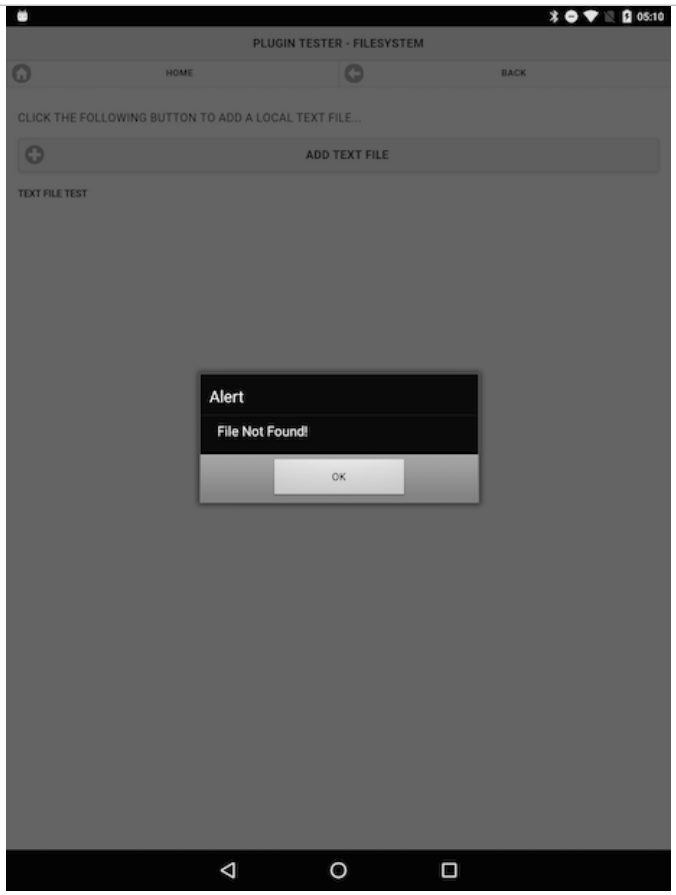
plugins - test filesystem `onFail()`

- 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 3

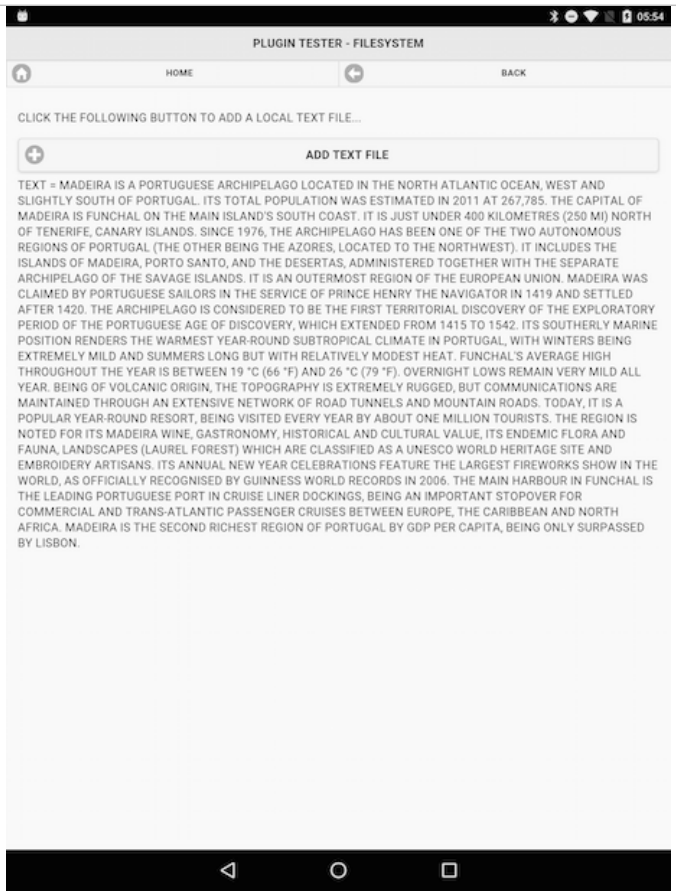
plugins - test filesystem with event

- 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 3

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 3

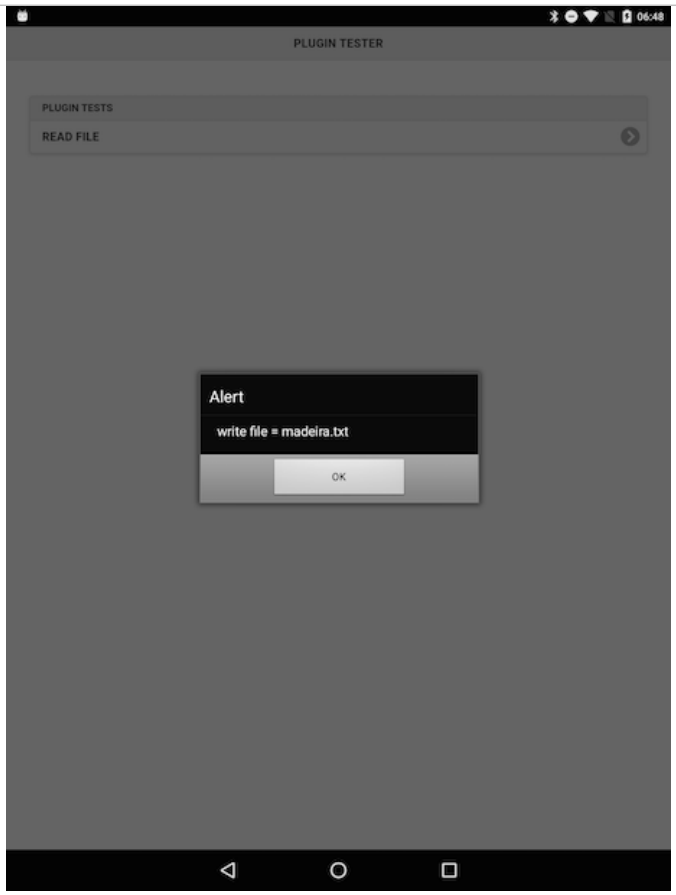
plugins - test filesystem with file write

- 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 3

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 3

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 3

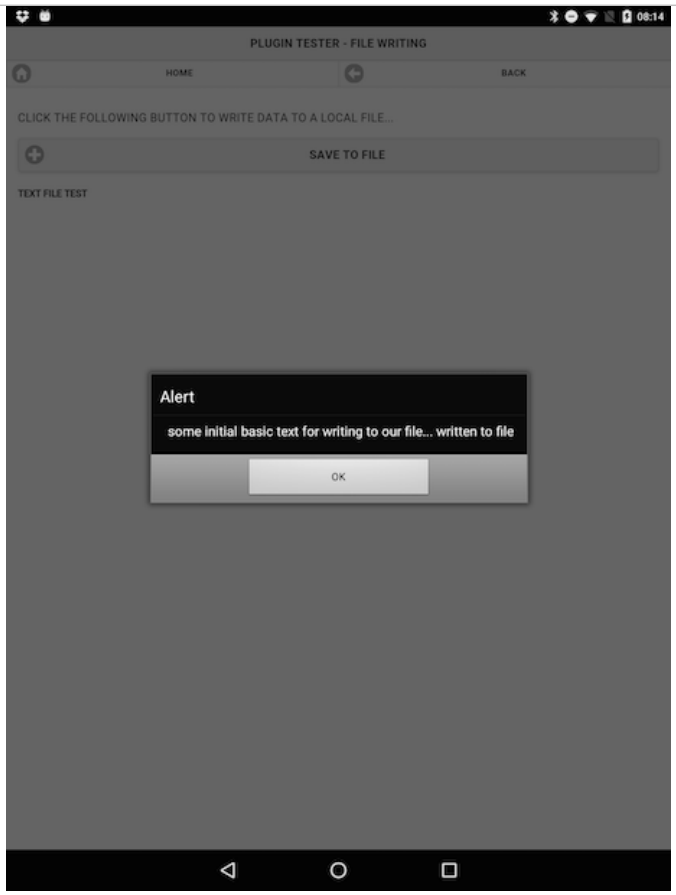
plugins - test filesystem with file write

- 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...");
});
```

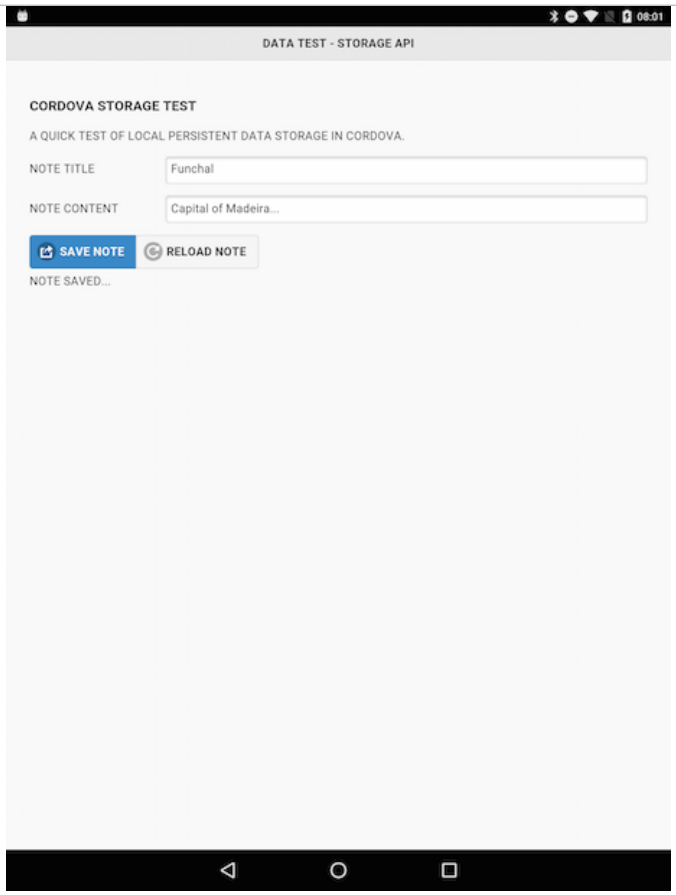
- could easily get text to write from an input field, from metadata...
- then pass it to our `writeTxtFile()` function for writing

Image - API Plugin Tester - file



API Plugin Tester - text written to file

Image - Data Tester



DataTest1 - save a note

Cordova app - LocalStorage - data test 1

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 1

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 1

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 1

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 1

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 1

app logic - storagenotes.js

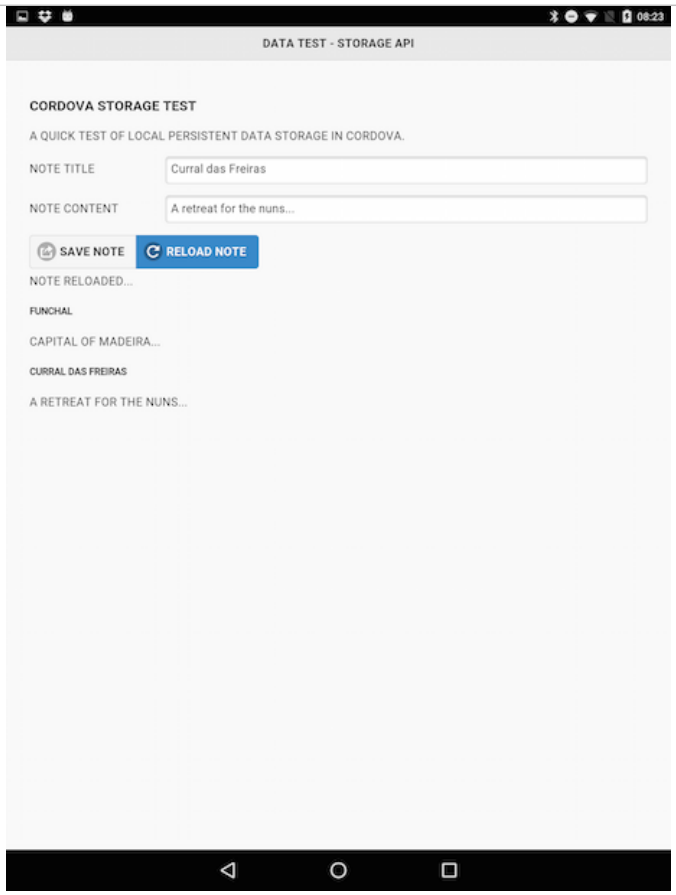
```
var NotesManager = (function () {
  var instance;

  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



DataTest1 - update the notes

Cordova app - API plugin examples - plugin test 4

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 4

plugins - geolocation

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

```
cordova create plugintestgeo com.example.plugintest PluginTestGeo
```

- add our required platforms for support and development,

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

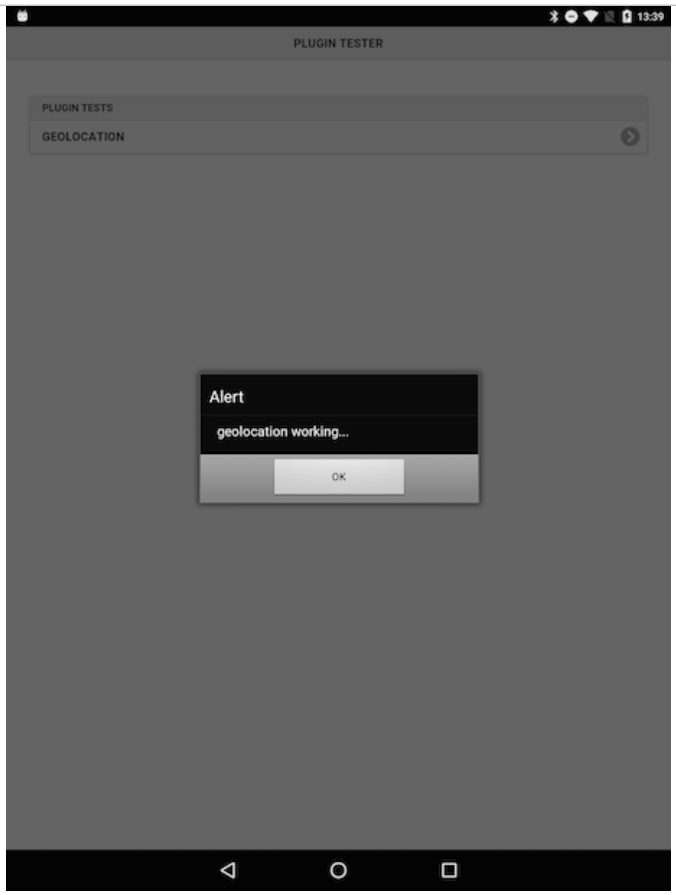
- update the `www` directory, modify initial settings in `config.xml`, and run initial test

```
//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 4

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*

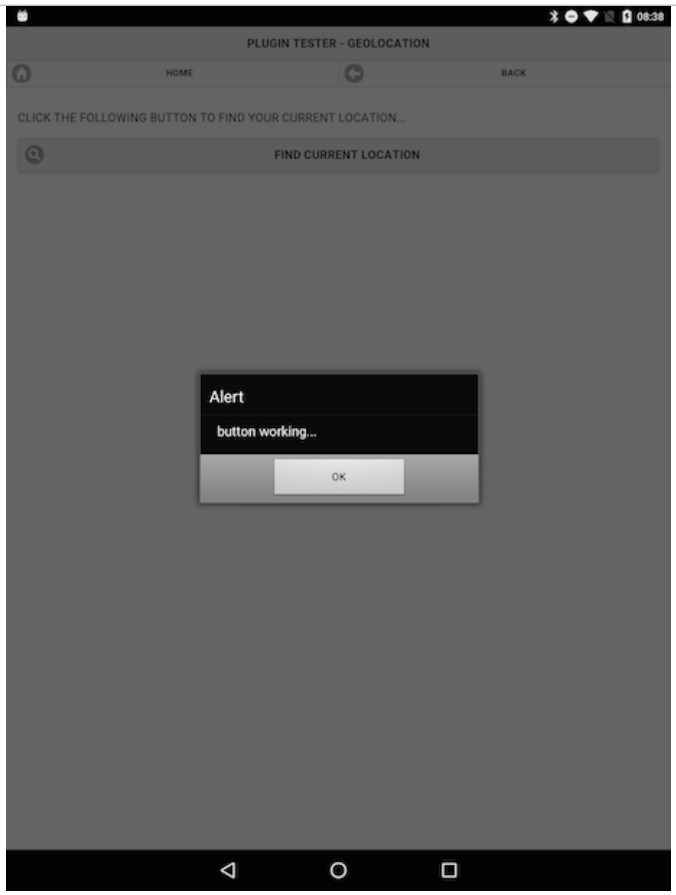
```
<div data-role="content">
  <p>Click the following button to find your current location...</p>
  <input type="button" id="getLocation" data-icon="search" value="Find Current Location" />
</div>
```

- 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 4

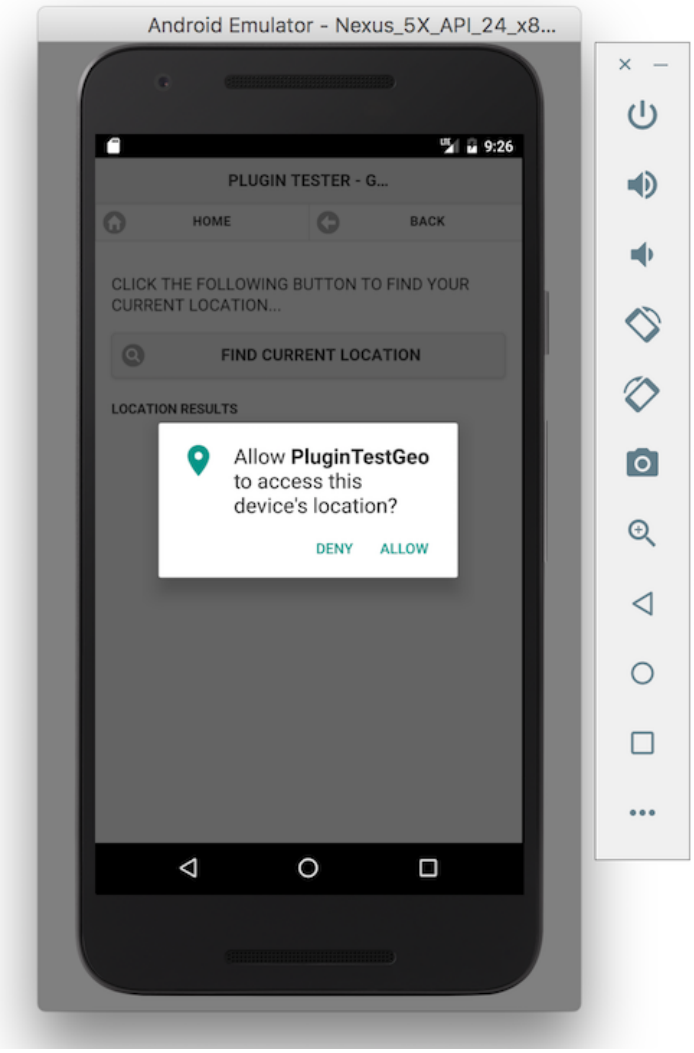
plugins - geolocation - test plugin

- 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 4

plugins - geolocation - test plugin

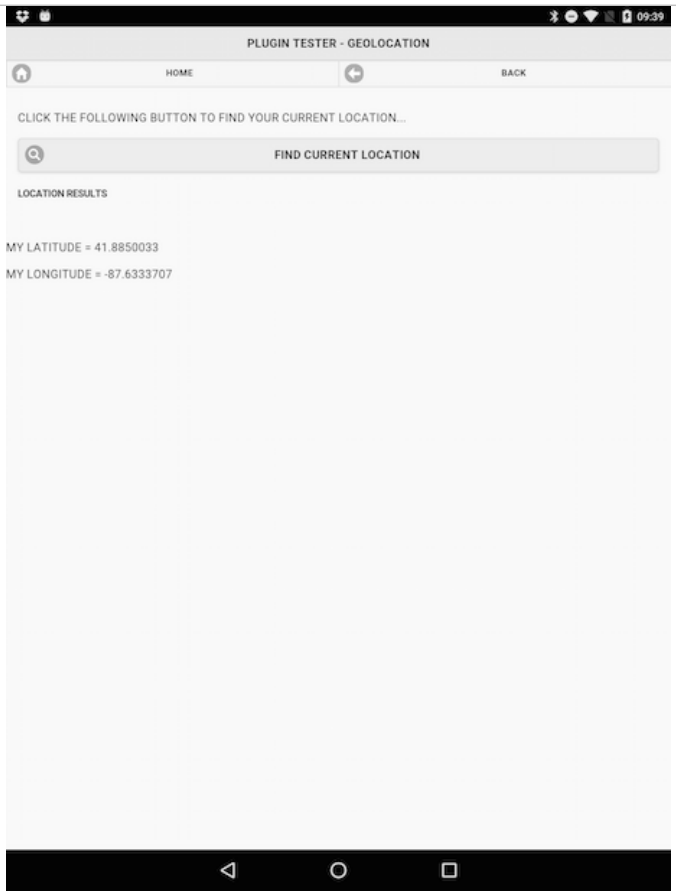
- successful return of location data
  - *use the latitude and longitude coordinates within our application*

```
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 4

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 4

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 4

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 4

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 4

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 4

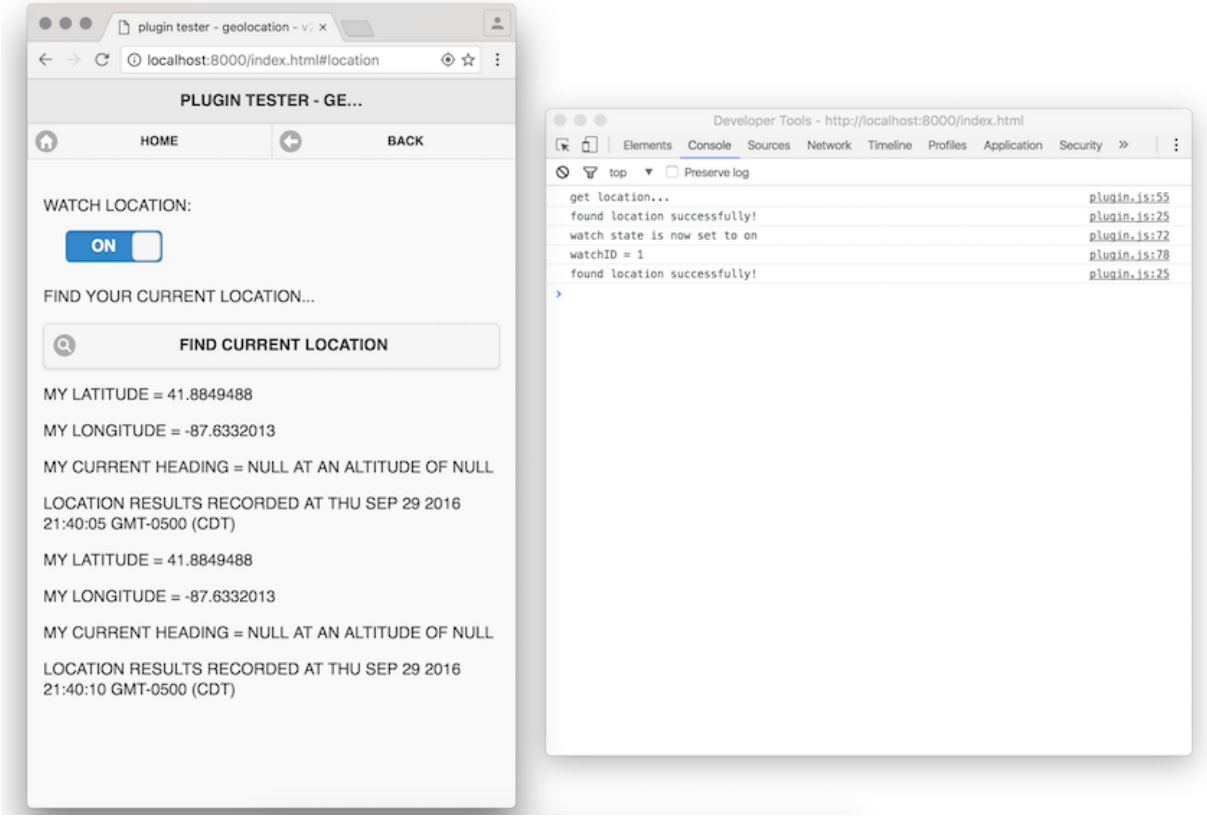
plugins - geolocation - manual toggle

- 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);
  }
});
```

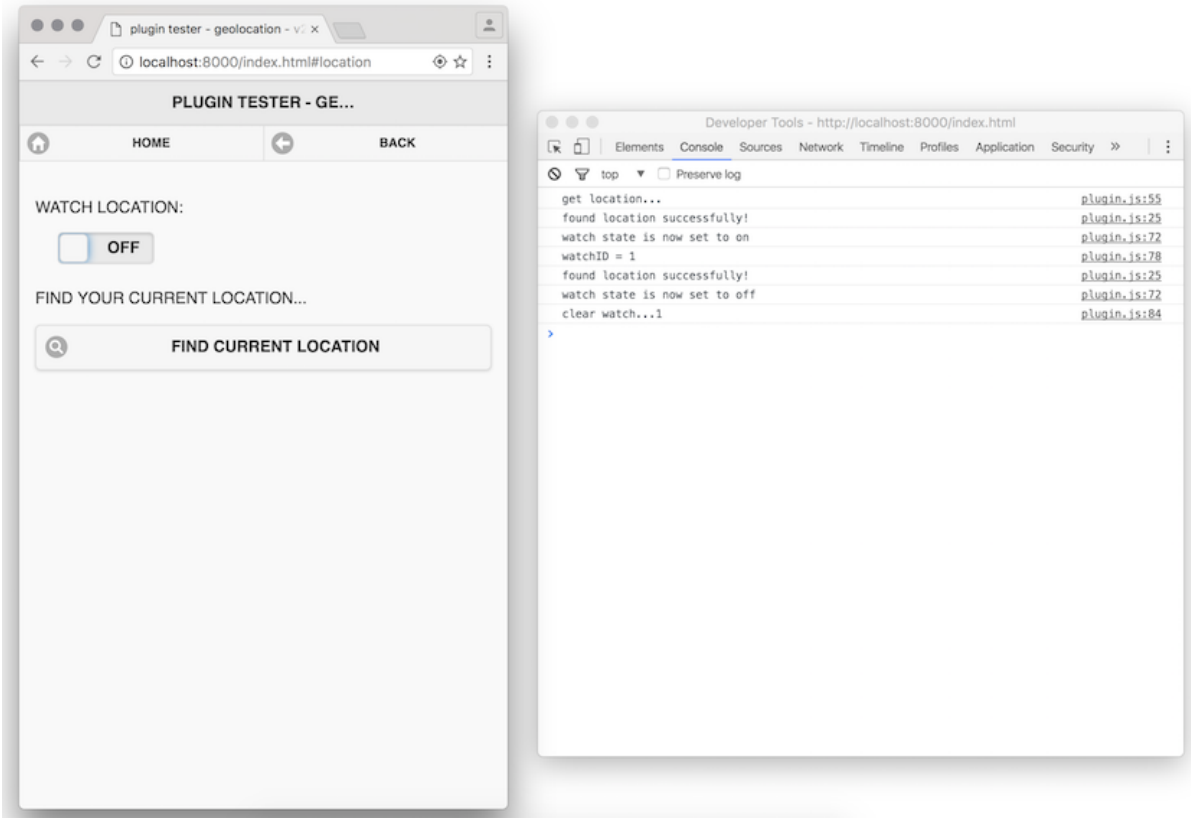
- update conditional statement
  - *clear output of coordinates, then clear watching of user's current location*

Image - API Plugin Tester - Geolocation toggle



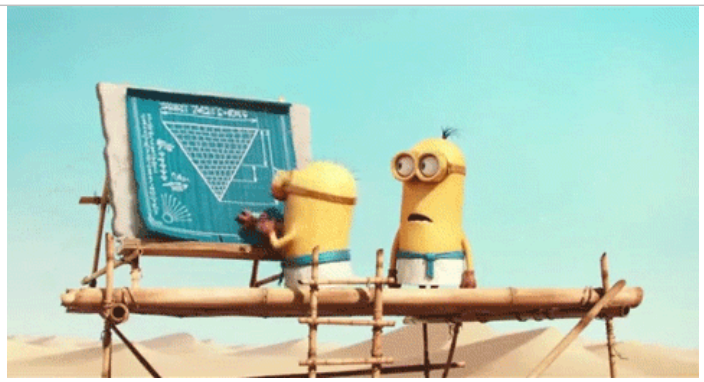
API Plugin Tester - geolocation toggle

Image - API Plugin Tester - Geolocation toggle



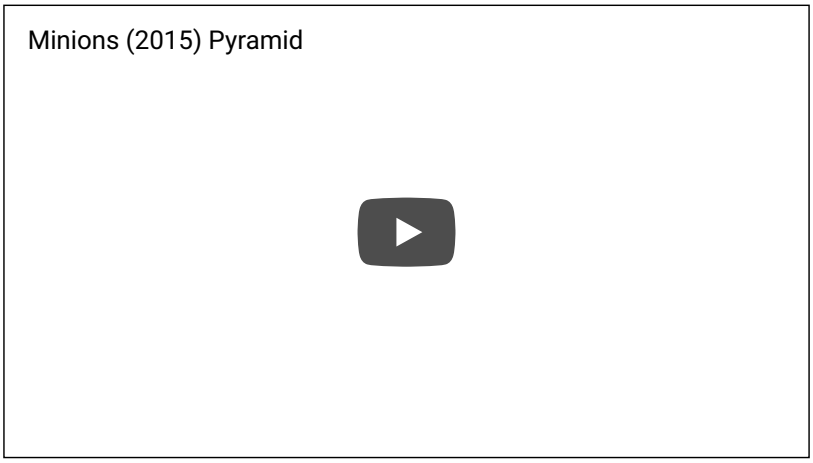
API Plugin Tester - geolocation toggle

Image - Designing our app



Designing our app - fundamentals are important

Video - Pyramid builders



Minions Pyramid Builders - Source: YouTube

References

- Cordova API
  - *config.xml*
  - *plugins*
  - *plugin - camera*
  - *plugin - filesystem*
  - *plugin - geolocation*
  - *Storage*
- Cordova Guide
  - *app templates*
- HTML5
  - *HTML5 File API*
- MDN
  - *Web APIs - FileError*
- OnsenUI
  - *OnsenUI v2*
  - *JavaScript Reference*
  - *Theme Roller*
- W3
  - *Web storage specification*
- W3C
  - *Geolocation API Specification*