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

Fall Semester 2018 - Week 2

Dr Nick Hayward

Cordova App - anatomy of a template - part 5

WWW directory

- three primary files initially help us develop Cordova application
  - *`index.html`*
  - *`index.js`*
  - *`index.css`*

Cordova App - anatomy of a template - part 6

- `index.html` - default template for new project

```html
<body>
    <div class="app">
        <h1>Apache Cordova</h1>
        <div id="deviceready" class="blink">
            <p class="event listening">Connecting to Device</p>
            <p class="event received">Device is Ready</p>
        </div>
    </div>
    <script type="text/javascript" src="cordova.js"></script>
    <script type="text/javascript" src="js/index.js"></script>
</body>
```

Cordova App - anatomy of a template - part 7

- default `index.html` page very straightforward
- `<div class="app">` is the parent section, acts as the app's container
- contains a child `div`
  - *unique ID `deviceready`*
  - *two key paragraphs triggered relative to state changes in the app*
- app simply updates state relative to event being actioned and listened
- events are monitored and controlled using the app's initial JavaScript
- `initialize()` method calls `bindEvents()` method
  - *adds an event listener to this `deviceready` div*
- means when device is ready `event listening` paragraph will be hidden
- `event received paragraph` is now shown

Cordova App - anatomy of a template - part 8

- js/index.js

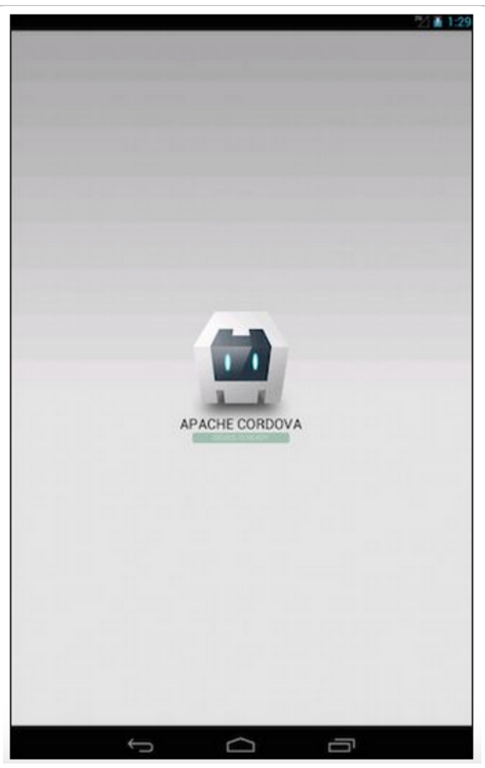
```
var app = {
    // Application Constructor
    initialize: function() {
        this.bindEvents();
    },
    // Bind Event Listeners
    //
    // Bind any events that are required on startup. Common events are:
    // 'load', 'deviceready', 'offline', and 'online'.
    bindEvents: function() {
        //document.addEventListener('deviceready', this.onDeviceReady, false);
        // update bind for ES6
        document.addEventListener('deviceready', (event) => this.onDeviceReady(event), false);
    },
    // deviceready Event Handler
    //
    // The scope of 'this' is the event. In order to call the 'receivedEvent'
    // function, we must explicitly call 'app.receivedEvent(...);'
    onDeviceReady: function() {
        app.receivedEvent('deviceready');
    },
    // Update DOM on a Received Event
    receivedEvent: function(id) {
        var parentElement = document.getElementById(id);
        var listeningElement = parentElement.querySelector('.listening');
        var receivedElement = parentElement.querySelector('.received');

        listeningElement.setAttribute('style', 'display:none;');
        receivedElement.setAttribute('style', 'display:block;');

        console.log('Received Event: ' + id);
    }
};

app.initialize();
```

Image - Cordova Splash Screen



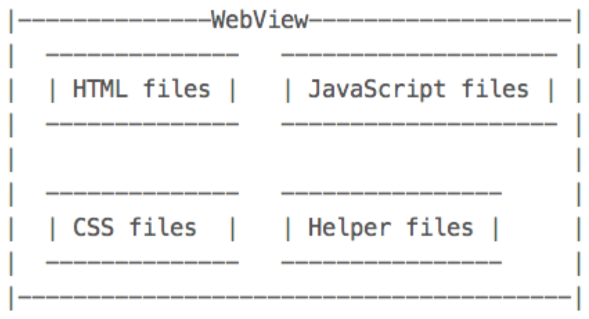
Apache Cordova Default Splashscreen

Apache Cordova - architecture - part 1

- Cordova relies on web technologies at its core
  - *HTML5*
  - *CSS*
  - *JavaScript (JS)*
- core architecture for app development using Cordova
- supplement this core with additional helper files
  - *e.g. JSON (JavaScript Object Notation) resource files*
- to enable access to a device's native functionality
  - *JS application objects (or functions) call Cordova APIs*
  - *Cordova APIs for different native mobile OSs, e.g.*
  - *use* Cordova Android *for native Android functionality...*
  - *use* Cordova iOS *for native iOS...*
- develop our own custom plugins as necessary

Image of Apache Cordova architecture

The following diagram summarises the core architecture for Cordova application development.



```
|--------------WebView-------------------|
|  --------------   -------------------- |
|  | HTML files |   | JavaScript files | |
|  --------------   -------------------- |
|                                        |
|  --------------   ----------------     |
|  | CSS files  |   | Helper files |     |
|  --------------   ----------------     |
|----------------------------------------|
```

Source - Apache Cordova

Apache Cordova - architecture - part 2

- core architecture creates a single screen in the native app
- single screen contains a **WebView**
- uses all of the device's available screen space (real estate)
- native WebView used to enable loading app's HTML, CSS, JS...
- WebView is a native view in each mobile OS
- allows us to display HTML based content
- allows us to leverage power and functionality of a mobile browser
- working within a contained native app

Apache Cordova - webview - part 1

- using this WebView in our app
- Cordova loads the app's default startup page
  - *in essence its `index.html` page*
- passes control of the app to the native WebView
- allows user to control the app as normal
- user can interact with app in native manner
- user gets a native app experience
- user interaction can include the vast majority of standard native interaction patterns and options
- user is not aware of difference between Cordova or native developed app

Apache Cordova - webview - part 2

- WebView has an implementation in all of the major mobile OSs
- Android has a class called

```
android.webkit.WebView
```

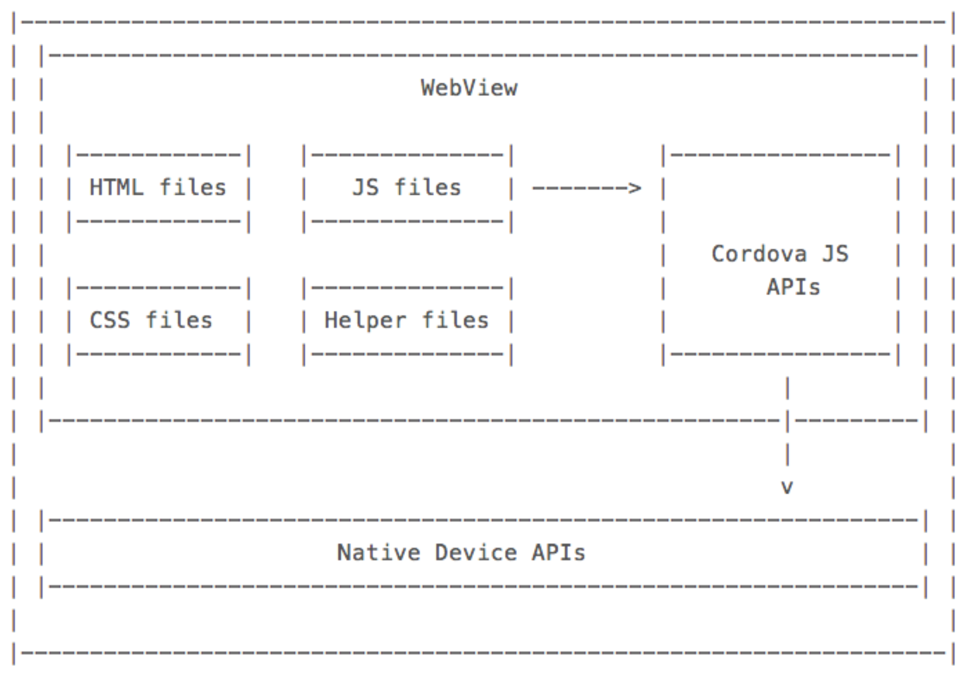
- iOS references the `UIWebView`
  - *part of the `UIKit` framework*
  - *n.b. from iOS 12 - `WKWebView` from `WebKit` API*
- Window Phone refers to a WebView class called

```
Windows.UI.Xaml.Controls
```

Apache Cordova - native functionality - part 1

- provides access to many types of native functionality, including
  - *sound and audio*
  - *recording*
  - *camera capture*
  - *photo access*
  - *geolocation*
  - *sensors...*
- Cordova leverages JavaScript APIs to provide native functionality

Image - Apache Cordova Native Functionality



Apache Cordova Architecture - JS APIs

Source - Apache Cordova

Apache Cordova - native functionality - part 2

- architecture is an elegant approach to solving cross-platform issues
- allows developers to leverage unified API interface
  - *perform specific native functions*
  - *calls to native functionality transparent across platforms*
    - strength of using JavaScript APIs
- Cordova JavaScript APIs
  - *call the required native OS API*
  - *e.g. Cordova's Android or iOS API*
- plugins give Cordova its power and flexibility

Apache Cordova - example call - part 1

If we want to get a picture from the camera, we call the following using Cordova

```
navigator.camera.getPicture(onSuccess, onFail, { quality: 75,
  destinationType: Camera.DestinationType.DATA_URL
});

function onSuccess(imageData) {
  var image = document.getElementById('Image');
  image.src = "data:image/jpeg;base64," + imageData;
}

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

Apache Cordova - example call - part 2

- making a simple call to the method `getPicture()` of the `camera` object
- call is performed with 3 parameters
- **onSuccess**
  - *callback allows us to tell the app what to do if the call and returned data is successful*
- **onFail**
  - *another callback tells the app how to handle an error or false return*
  - *e.g. an error is thrown, callback will handle output of a suitable error message*
- **quality**

Apache Cordova - example call - part 3

```
quality: 75, destinationType: Camera.DestinationType.DATA_URL
```

- slightly different as it contains a JS object with configuration parameters
- two parameters are for `quality` and `destinationType`
- `quality` can be from `0 to 100`
- `destinationType` refers to the required format for the returned data value
  - *can be set to one of 3 possible values*
  - *`DATA_URL` - format of the returned image will be a Base64 encoded string*
  - *`FILE_URL` - returns the image file URL*
  - *`NATIVE_URI` - refers to the images native URI*

Apache Cordova - example call - part 4

- if the return is a success we will get a Base64 encoded string
  - *string of the image just captured using the native camera*
- leveraging the power of the Apache Cordova camera plugin code, e.g. Android camera plugin
- power of the underlying Android class
  - *wrapped in a layer that we can call from our JavaScript code*
- plugin is written natively for Android
  - *we access it using JS with Cordova*
- plugins for other platforms follow the same pattern
  - *e.g. iOS camera plugin...*

Apache Cordova - example call - part 5

- we issue a call from JS using Cordova to the native code in the plugin
- plugin processes this request
  - *returns the appropriate value*
  - *either for a success or a failure*
- in our example, if request to the camera is successful
  - *Android plugin will return a string to the JS Cordova client, as requested*
- use similar pattern for other mobile OSs
  - *e.g. accessing a camera's functionality with iOS...*
  - *appropriate plugin required for necessary mobile OS*
  - *if not, we can write a custom plugin*

Apache Cordova - cross-platform power

- implement capturing a photo from device's native camera on multiple mobile platforms
- Cordova plugin architecture removes
  - *need to understand how the photo capture is implemented or handled natively*
- Cordova plugin handles the native calls
- Cordova plugin handles processing for each native device

Cordova - CLI - Useful commands

A few initial useful CLI commands

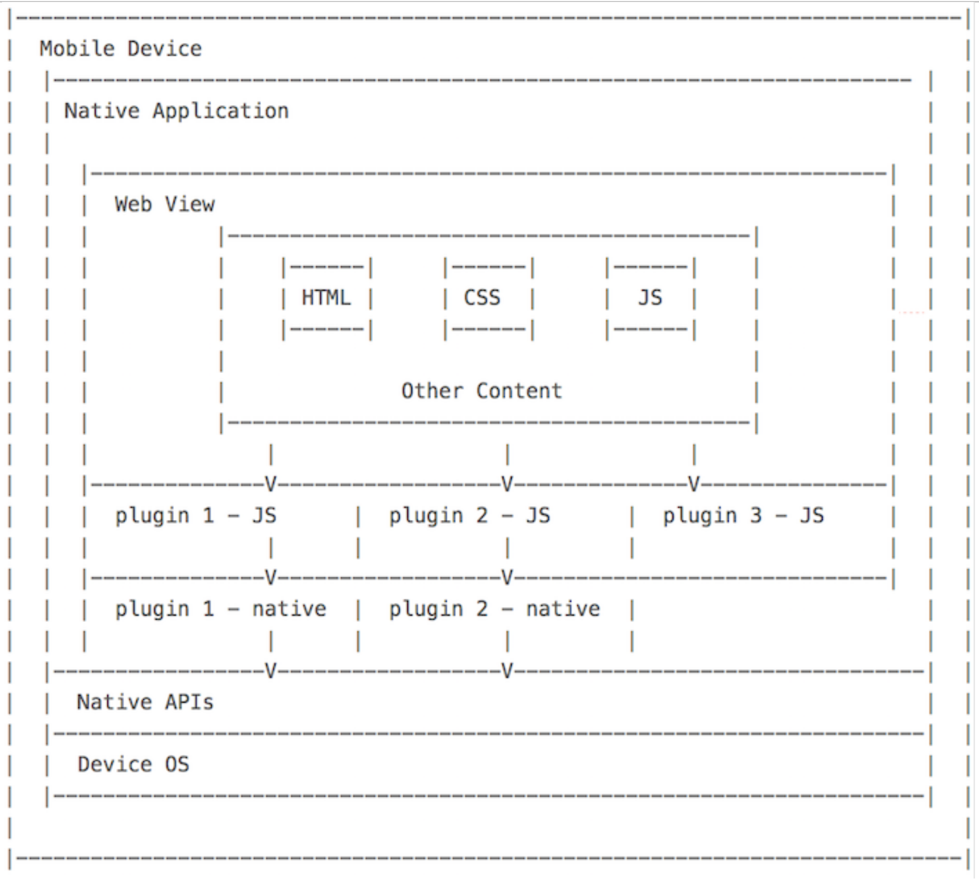
command	example	description
cordova	cordova	general command - outputs overview with 5 categories of information and help
-v	cordova -v	check current installed version of cordova
requirements	cordova requirements	check requirements for each installed platform
create	cordova create basic com.example.basic 422Basic	creates new project with additional arguments for directory name, domain-style identifier, and the app's display title
platform add	cordova platform add android --save	specify target platforms, eg: Android, iOS... (NB: SDK support required on local machine)
platform ls	cordova platform ls	checks current platforms for cordova development on local machine and lists those available
platform remove (platform rm)	cordova platform rm android	remove an existing platform
build	cordova build	iteratively builds the project for the available platforms
build ios	cordova build ios	limit scope of build to a specific platform (useful for testing a single platform...)
prepare	cordova prepare ios	prepare a project, and then open and build &c. with native IDE (eg: XCode, Android Studio...)
compile	cordova compile ios	compile ios specific version of app
emulate	cordova emulate android	rebuilds an app and then launches it in a specific platform's emulator
run	cordova run android	run an app on a native device connected to the local machine
run --list	cordova run --list	check available emulators, e.g. Android AVDs

- more commands will be added as we work with Cordova, NPM...

Cordova Design - architecture - intro

- quickly recap the architecture and design behind a Cordova Native application
- Cordova effectively consists of the following components
  - *source code to allow us to build a native application container*
  - *specific to the mobile platforms we choose to add to our project, eg: Android, iOS...*
  - *a collection of various APIs, implemented by Cordova as plugins*
  - *web application running within the container*
  - *access to native device functionality, APIs, and applications*
  - *provides a useful set of tools that help us manage our projects*
  - *creating a project, project files...*
  - *manage required plugins*
  - *build native applications using the native SDK*
  - *testing of applications using emulators, simulators...*

Cordova Design - architecture - diagram



Cordova - Architecture

Cordova Design - architecture

JS & Web plugins

- outline architecture includes the option for JavaScript only plugins
- JS plugins in Cordova normally a bridge from our web container to the native APIs
  - *useful way to expose native device functionality to the web application*
- use and develop plugins purely in JS
  - *add an existing library to help with data visualisations, graphics...*
- create our own focused plugins
  - *abstraction of application features and logic, other specific requirements...*
- greater support for native functionality at the web application level
- HTML5 APIs

Cordova Design - architecture - web container - part 1

- Cordova development
  - *uses many of the same underlying technologies as standard web application development*
  - *a few limitations relative to network access that we need to consider*
- hybrid mobile application with Cordova
  - *a web application needs to be written as a self-contained application*
  - *needs to be able to run within web container on native device*
  - *constantly fetching external resources not good practice*
  - *mix of local and remote resources preferable for most apps*
  - *external resources an issue if we lose a network connection*
- `index.html` file will normally be the only HTML file we use
  - *separate pages will be containers within this file*

Cordova Design - architecture - web container - part 2

- rethink our approach to building such mobile web stack applications
  - *help us leverage the inherent capabilities of Cordova*
- self-contained applications need to ensure
  - *any application files and data are initially available*
  - *allows the application to launch and load on the native device*
  - *without initial calls to a remote server*
  - *load the application and render the UI*
- application can then optionally fetch data
  - *remote server, API, search query, stream media...*
- consider stages of design for our app's container

Cordova Design - architecture - SDKs and OSs

- build our Cordova applications
  - *including default Cordova APIs or additional APIs*
  - *each app has to be packaged into a native application*
  - *allows app to run on the host native device*
- each native SDK has its own set of custom or proprietary tools
  - *building and packaging their specific native applications*
- build our Cordova applications for a native device
  - *web content portion of app is added to a project*
  - *applicable to the chosen mobile platforms,*
  - *e.g. Android, iOS, Windows 10 Universal Platform...*
  - *project is then built for each required platform*
  - *using Cordova CLI, for example*
  - *uses each of the applicable platform specific set of tools to help build*

Cordova App - CLI recap

build initial project

```
cd /Users/ancientlives/Development/cordova
cordova create basic com.example.basic Basic
cd basic
```

- creates new project ready for development

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

- adds support for native SDK, Android
- then builds the project ready for testing and use on native device

```
cordova emulate android
```

- outputs current project app for testing on Android emulator

```
cordova prepare android
```

- copies app code into platform ready for building
  - *then use native IDE for build &c...*

Cordova App - structure recap - app directory

- quick recap of app's structure
- new project includes the following default structure

```
|- config.xml
|- hooks
|- package.json
|- README.md
|- platforms
   |- android
   |- platforms.json
|- plugins
|  |- android.json
|  |- cordova-plugin-whitelist
|  |- fetch.json
|- res
|  |- icon
|  |- screen
|- www
|  |- css
|  |- img
|  |- index.html
|  |- js
```

- initially, our main focus will be the www directory

Cordova App - structure recap - www directory

```
|- www
|  |- css
|     |- index.css
|  |- img
|     |- logo.png
|  |- index.html
|  |- js
|     |- index.js
```

Cordova App - basics of development - part 1

default `index.html`

```html
<html>
    <head>
        <meta http-equiv="Content-Security-Policy" content="default-src 'self'
        data: gap: https://ssl.gstatic.com 'unsafe-eval'; style-src 'self'
        'unsafe-inline'; media-src *">
        <meta name="format-detection" content="telephone=no">
        <meta name="msapplication-tap-highlight" content="no">
        <meta name="viewport" content="user-scalable=no, initial-scale=1,
        maximum-scale=1, minimum-scale=1, width=device-width">
        <link rel="stylesheet" type="text/css" href="css/index.css">
        <title>Hello World</title>
    </head>
    <body>
        <div class="app">
            <h1>Apache Cordova</h1>
            <div id="deviceready" class="blink">
                <p class="event listening">Connecting to Device</p>
                <p class="event received">Device is Ready</p>
            </div>
        </div>
        <script type="text/javascript" src="cordova.js"></script>
        <script type="text/javascript" src="js/index.js"></script>
    </body>
</html>
```

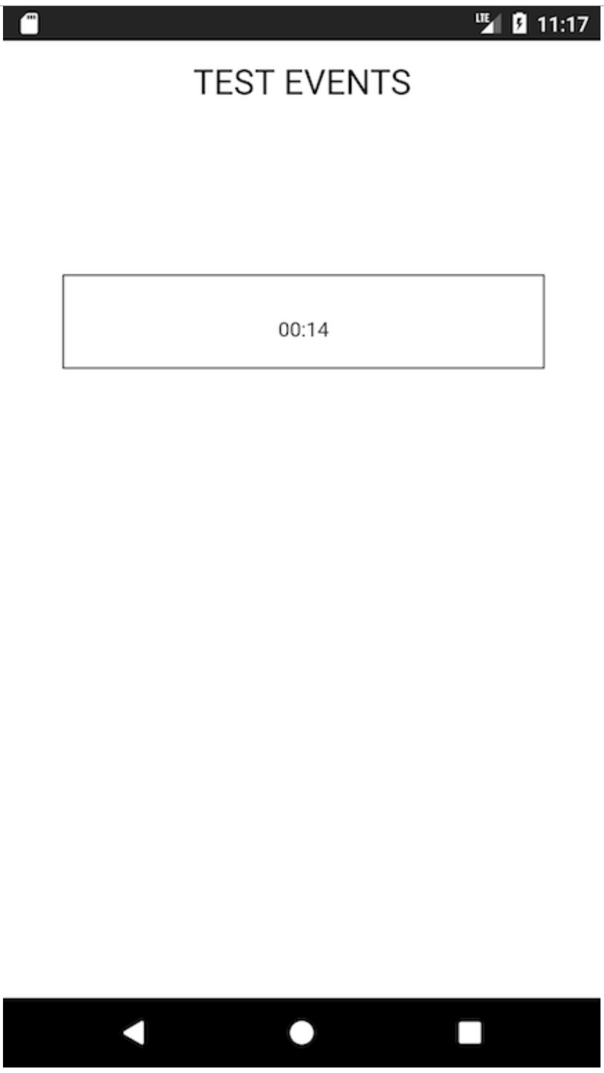
Cordova App - basics of development - part 2

test app `index.html`

```html
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Security-Policy" content="default-src 'self' data: gap: https://ssl.gstatic.co
    <meta name="format-detection" content="telephone=no">
    <meta name="msapplication-tap-highlight" content="no">
    <meta name="viewport" content="user-scalable=no, initial-scale=1, maximum-scale=1, minimum-scale=1, widt
    <link rel="stylesheet" type="text/css" href="css/index.css">
    <title>Basic Events</title>
  </head>
  <body>
    <main>
      <header>
        <h3>Test Events</h3>
      </header>
      <section id="events">
        <!-- output current status relative to PAUSE event... -->
        <p id="pause"></p>
        <!-- output current status relative to RESUME event... -->
        <p id="resume"></p>
        <!-- output timer to check loading and app events -->
        <div id="timer">
          <label id="minutes">00</label>:<label id="seconds">00</label>
        </div>
      </section>
    </main>
    <!-- load JS files for app - cordova.js required -->
    <script type="text/javascript" src="cordova.js"></script>
    <!-- load app main file -->
    <script type="text/javascript" src="js/index.js"></script>
  </body>
</html>
```

- app structure using HTML5 semantic structure
- lack of styling will be an issue...

Image - Cordova App - Basic Events



Basic Events

Cordova App - basics of development - part 3

add Cordova specifics

- Cordova container for the application
  - *exposes native APIs to web application running in WebView*
- most APIs not available until applicable plugin added to the project
- container also needs to perform some preparation before the APIs can be used
- Cordova informs us when the container, and associated APIs, are ready for use
- fires a specific event, called the `deviceready` event
- application logic requiring use of Cordova APIs
  - *should be executed after receipt of `deviceready` notification*

Cordova App - basics of development - part 4

check `deviceready` event

```
/*
* FN: loader for the main app
* - check deviceready event
* - bootstrap app loading & events
*/
function onLoad() {
  // Add the deviceready event
  document.addEventListener("deviceready", function(){

  // attach test events
  document.addEventListener("pause", onPause, false); // pause event
  document.addEventListener("resume", onResume, false); // resume event

  // start test timer
  testTimer();

  }, false);

}

// LOADER - load app & check for deviceready event...
onLoad();
```

- updated loader function for app...
- add test events for `pause` and `resume`
  - *useful for Android...*
- Cordova Docs - Events

Cordova App - basics of development - part 5

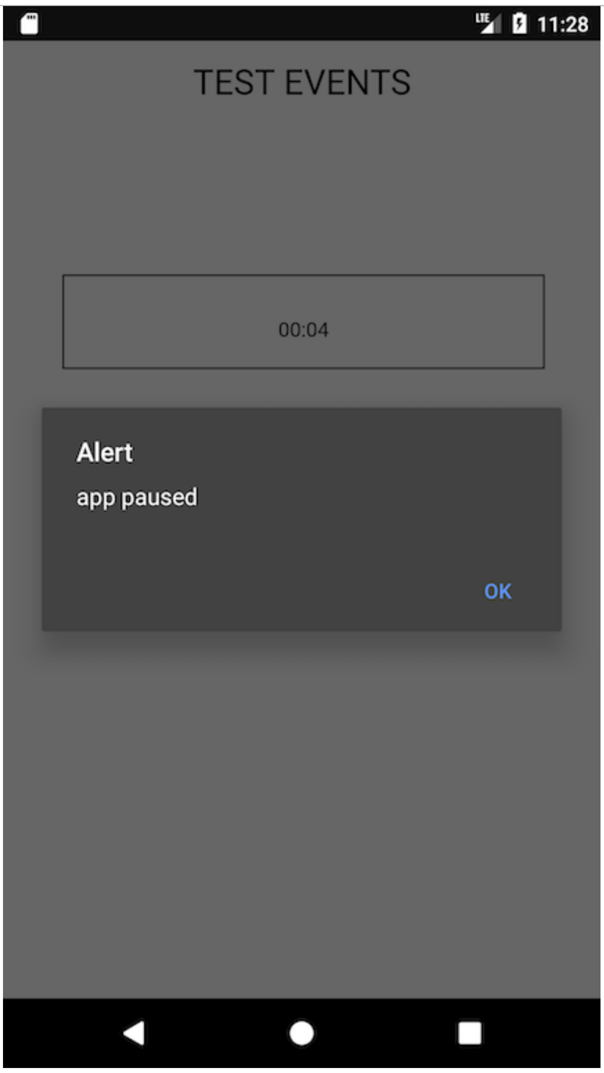
respond to events - pause

- pause

```
// FN: call in response to Pause event
function onPause() {
    // get current Unix timestamp
    const currentTime = Date.now();
    // get status element in DOM
    const pause = document.getElementById('pause');
    // create text node to update DOM
    const text = document.createTextNode(`app has been paused...${currentTime}`);
    // append text to status element
    pause.appendChild(text);
    // show alert in native UI
    alert('app paused');
}
```

Image - Cordova App - Basic Events

Pause



Alert

app paused

Basic Events - Pause

Cordova App - basics of development - part 6

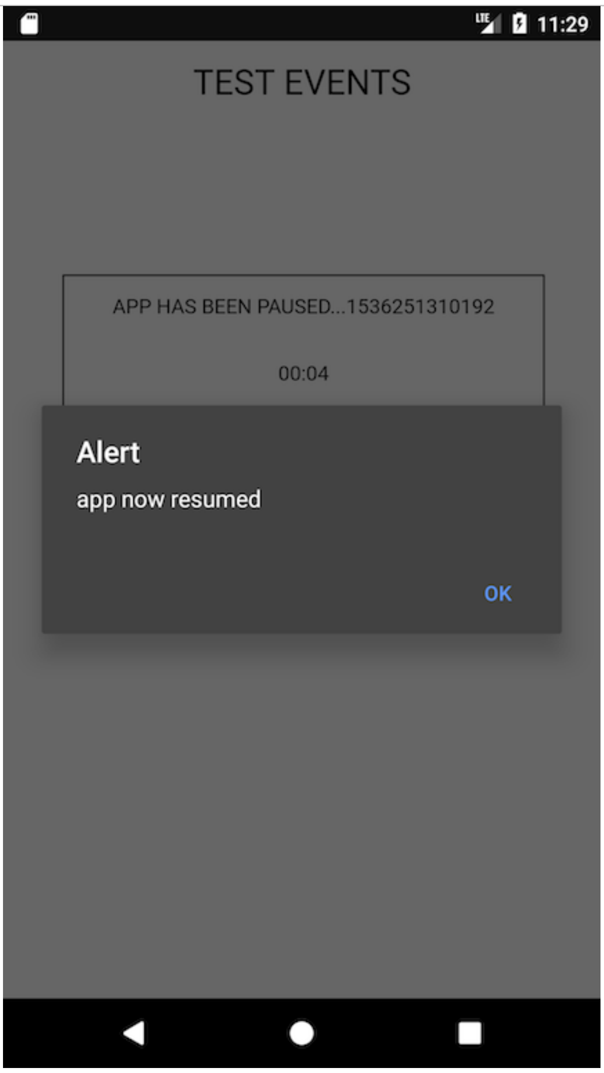
respond to events - resume

- `resume`

```
// FN: call in response to Resume event
function onResume() {
    // get status element in DOM
    const resume = document.getElementById('resume');
    // create text for output
    const text = document.createTextNode("app has been resumed...");
    // append text to status element
    resume.appendChild(text);
    // show alert in native UI
    alert('app now resumed');
}
```

Image - Cordova App - Basic Events

Resume



TEST EVENTS

APP HAS BEEN PAUSED...1536251310192

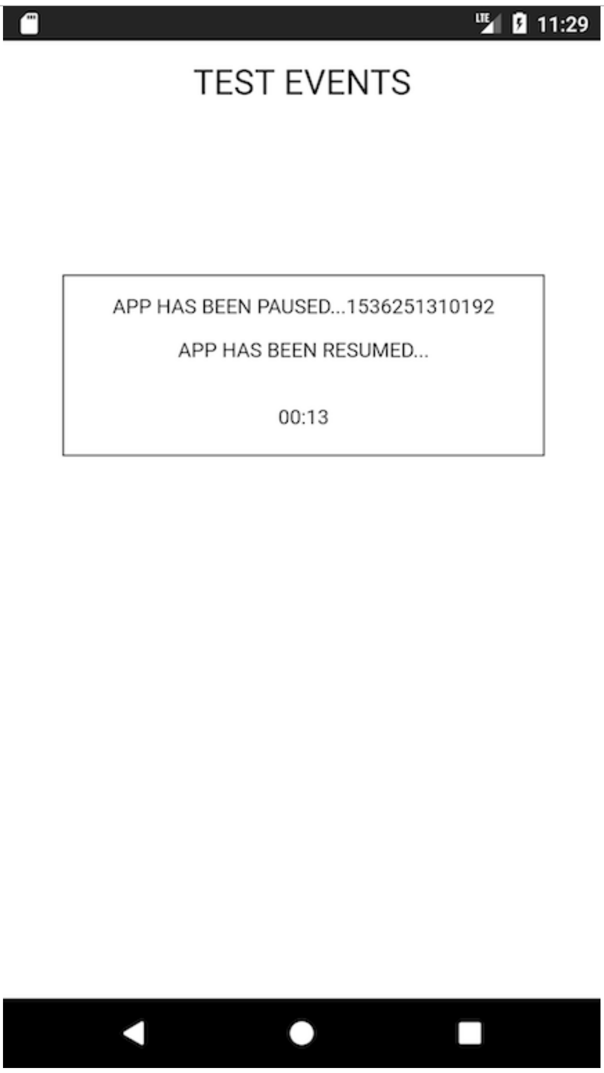
00:04

Alert

app now resumed

Basic Events - Resume

Image - Cordova App - Basic Events


11:29

TEST EVENTS

APP HAS BEEN PAUSED...1536251310192

APP HAS BEEN RESUMED...

00:13

Basic Events

Cordova app - working with plugins - getting started

- start looking at some of the plugins available for Cordova
  - *media playback &c.*
- test our initial design and structure
  - *add some existing plugins*
  - *see how they fit together to create a coherent, basic application*
- create our new project

```
cordova create plugintest1 com.example.plugintest plugintest1
```

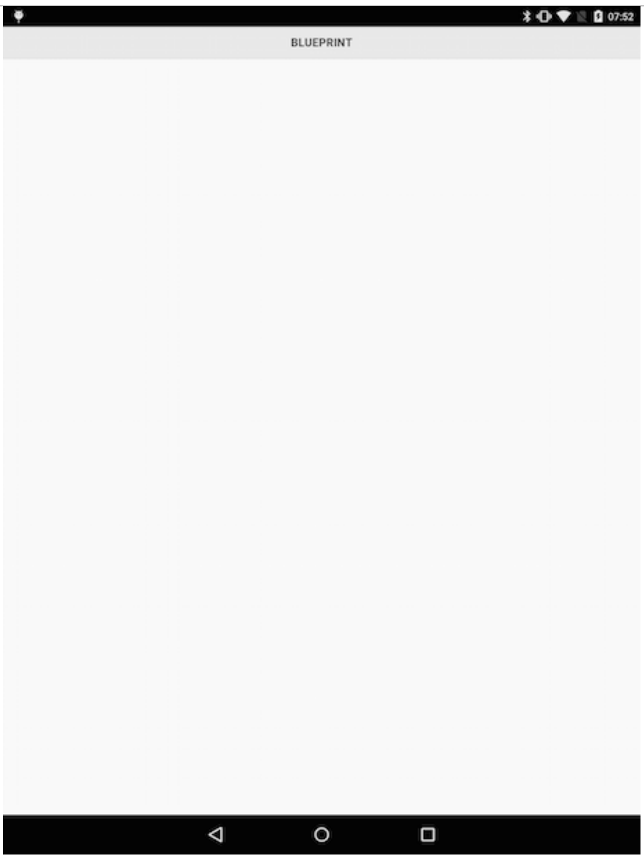
- add support for Android platform

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

- add support for other platforms, as required, such as iOS, Windows...
- transfer our default www directory
- start updating some of the settings in the `config.xml` file for the application
  - *metadata for author, description, name...*
- quickly run and test this base for our new application

```
//run in the Android emulator
cordova emulate android
//run on a connected Android device
cordova run android
```

Image - Cordova app - Plugin Test 1 - getting started



Cordova - Plugin Test - getting started

Cordova app - working with plugins - add plugins

- add our required plugins to the test application
  - *add plugins for **device**, **file**, and **media***
- **device** plugin added to check and read information about current device
  - *in effect our Android phone or tablet*
- **file** plugin is required to access the device's underlying filesystem
- **media** helps us record and playback media files
- add these plugins to our project with the following Cordova commands

```
//add device plugin - Git and NPM options
cordova plugin add https://git-wip-us.apache.org/repos/asf/cordova-plugin-device.git
cordova plugin add cordova-plugin-device
//add file plugin - Git and NPM options
cordova plugin add https://git-wip-us.apache.org/repos/asf/cordova-plugin-file.git
cordova plugin add cordova-plugin-file
//add media plugin - Git and NPM options
cordova plugin add https://git-wip-us.apache.org/repos/asf/cordova-plugin-media.git
cordova plugin add cordova-plugin-media
```

- ensure new plugins are applied to our current project
  - *run the following Cordova command*

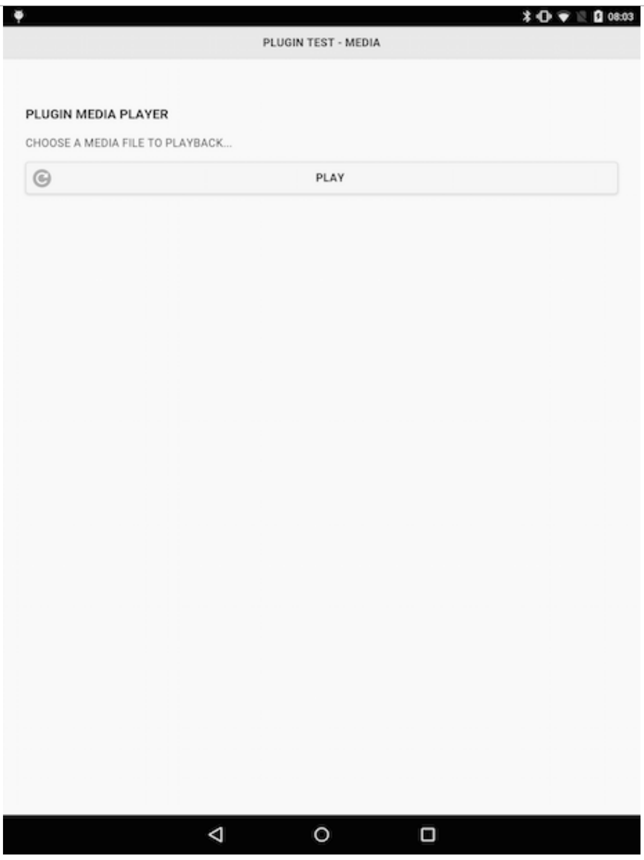
```
cordova build
```

n.b. *NPM plugin install is now recommended for latest Cordova apps*

Cordova app - working with plugins - update `index.html`

- update our `index.html` page to create the basic layout
  - *allow us to load and use media files*
- use a single page application structure
  - *include our content categories for `header`, `main` &c.*
- add specific nodes for app structure
  - *signifies that we have a contiguous group of form, input elements &c.*
- use this grouping to add our **play** button
  - *load our sample file using the installed plugins*
  - *perhaps add an icon for the playback option*

Image - Cordova app - Plugin Test 1 - getting started



Cordova - Plugin Test - index.html

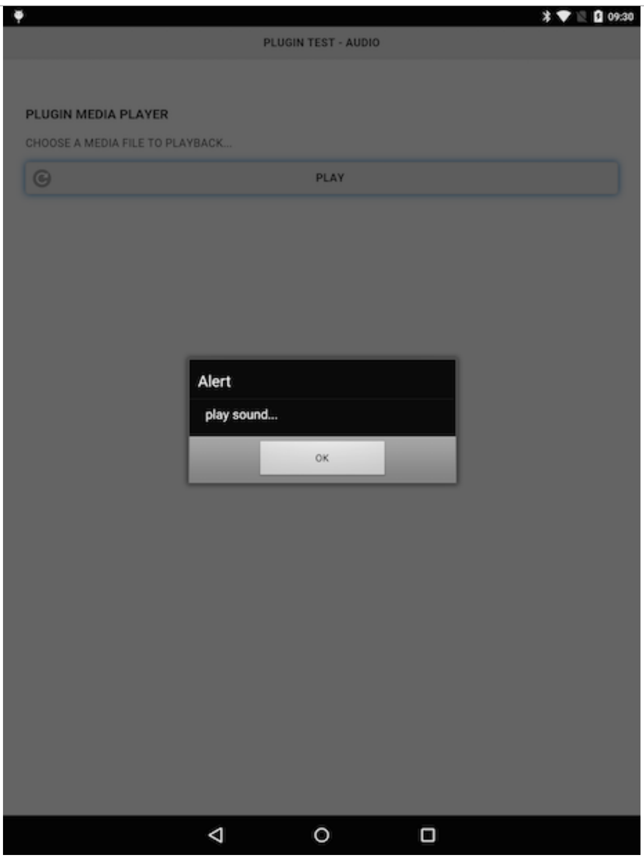
Cordova app - working with plugins - add some logic

- add some logic to our application
- updates to our JavaScript to allow us to handle events
- add handlers for listeners for each button we add to the application
  - *including the initial **play** button*
- add this code to our application's custom JavaScript file
  - *`plugin.js`*
- setup the application in response to Cordova's `deviceready` event
  - *event informs us that installed plugins are loaded and ready for use*
- add a function for the `deviceready` event
  - *allows us to bind our handler for the tap listener on the **play** button*

Cordova app - working with plugins - `onDeviceReady()`

- add any other required, initial functions later to this same start-up function
- wrap initial function in our main application loader
  - *checks device is ready, and then adds any required handlers*
- handlers required for audio, e.g.
  - *play*
  - *pause*
  - *stop*
  - *record*
  - ...

Image - Cordova app - Plugin Test 1 - getting started



Cordova - Plugin Test - audio button

Cordova app - working with plugins - audio playback logic

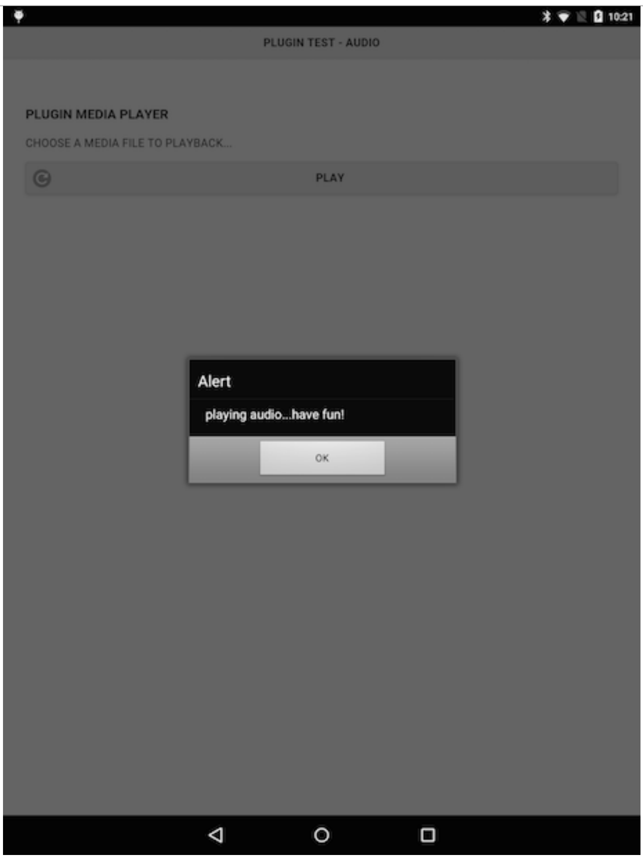
- now setup and tested the basic app logic
  - *added handlers for `deviceready` and clicking the audio playback button*
- update logic for the `#playAudio` button

```
//play audio file
function playAudio() {
  //initial url relative to WWW directory - then built for Android
  var $audioURL = buildURL("media/audio/egypt.mp3");
  var $audio = new Media($audioURL, null, errorReport);
  $audio.play();
  alert("playing audio...have fun!");
}
```

- add associated media loaders for the audio file
- add basic error checks in case the media file is missing, corrupt...

```
//build url for android
function buildURL(file) {
  if (device.platform.toLowerCase() === "android") {
    var $androidFile = "/android_asset/www/" + file;
    return $androidFile;
  }
}
//return any error message from media playback
function errorReport(error) {
  alert("Error with Audio - " + JSON.stringify(error));
}
```

Image - Cordova app - Plugin Test 1 - getting started



playing audio...have fun!

Cordova - Plugin Test - audio playback

Cordova app - working with plugins - update media playback

- basic plugin test for media playback within an app
  - *user can play music in their app*
  - *user touch interaction with button*
  - *file loaded from local filesystem*
  - *device playback of selected audio file*
- leveraging native device functionality in app
  - *calling plugins for **device**, **file**, **media**...*
- basic app includes,
  - *user interaction in the UI*
  - *calls to the exposed JS API for the plugins*
  - *playback of audio by the native device*
- add further functionality
  - *stop, pause...*

Cordova app - working with plugins - stop button

- consider how to **stop**, **pause** playback
  - *e.g. UI interaction, timer, event...*
- app logic is very similar
  - *respond to **stop** event*
  - *call method*
  - ...
- methods for **stop**, **pause**, &c. available in plugin API

```
media.pause
media.stop
media.release
```

Cordova app - working with plugins - stop button - part 1

- start to update our existing app by adding a **stop** button to the UI
  - *allow our user to simply tap a button to stop playback*
- update initial JS logic for the app
  - *listen for tap event on **stop** button*
  - *then call the stop method on the **media** object*

Cordova app - working with plugins - stop button - part 2

- add the logic for our custom method to stop the audio
  - *call as `stopAudio()`*

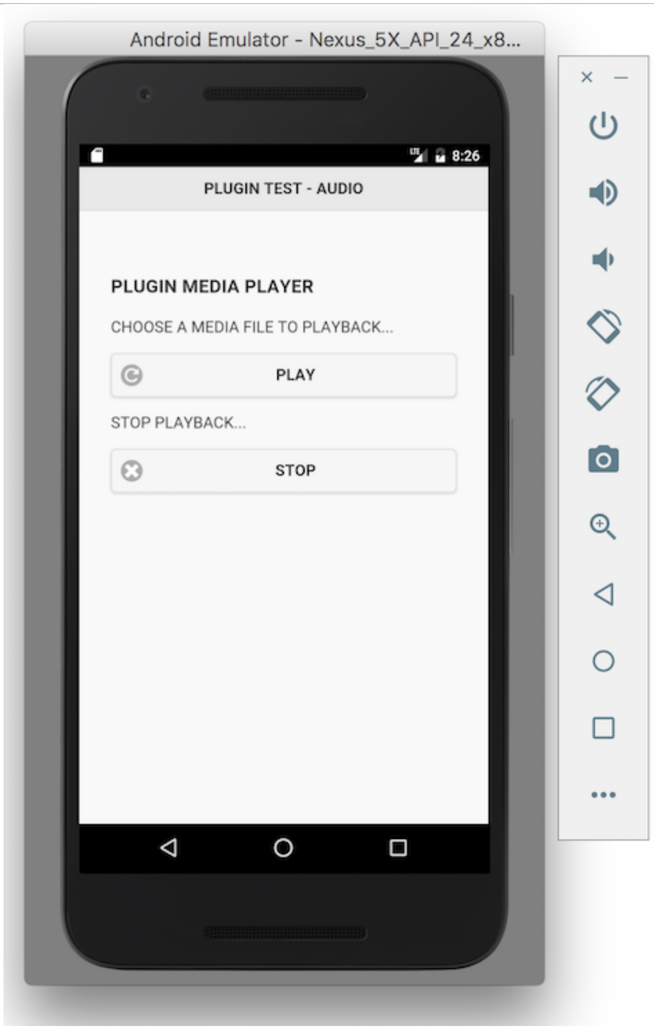
```
//stop audio file
function stopAudio() {
    //stop audio playback
    $audio.stop();
    //release audio - important for android resources...
    $audio.release();
    //just for testing
    alert("stop playing audio...& release!");
}
```

- logic still won't stop the audio playing
- issue is variable `$audio`
  - *currently restricted local scope to `playAudio()` method*
- initially alter scope of property for `$audio` itself
  - *now set in initial `onDeviceReady()` method*

```
function onDeviceReady() {
    //set initial properties
    var $audio;
...
}
```

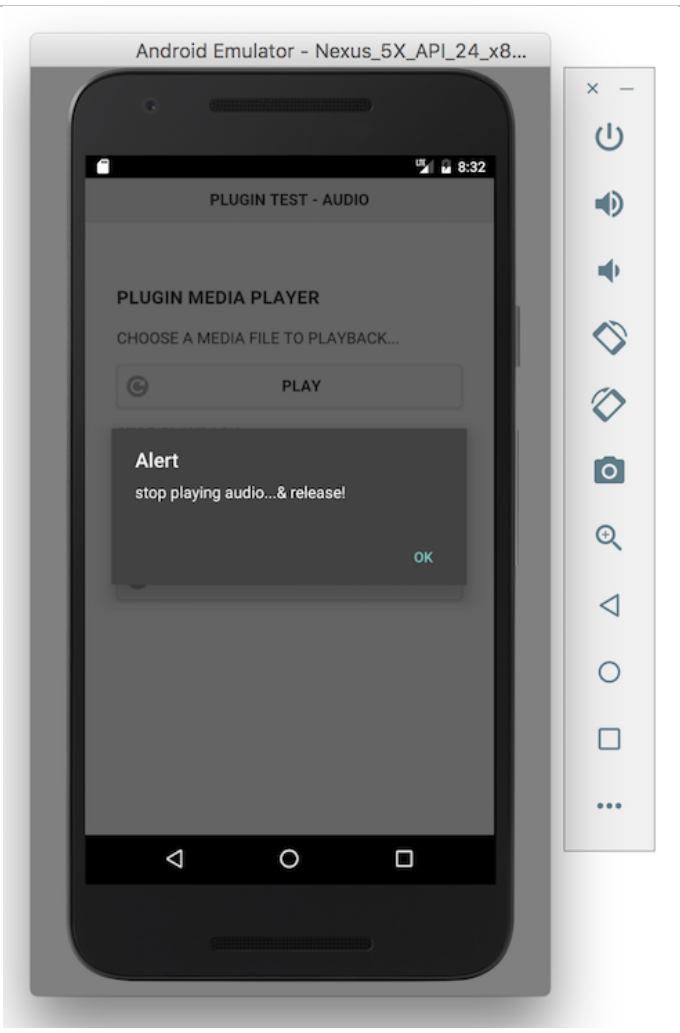
- logic will now stop audio playing
- call to `release()` method important for OS's audio resources
  - *particularly important to release unwanted resources on Android...*

Image - Cordova app - Plugin Test - stop audio playback


Android Emulator - Nexus_5X_API_24_x8...
8:26
PLUGIN TEST - AUDIO
PLUGIN MEDIA PLAYER
CHOOSE A MEDIA FILE TO PLAYBACK...
PLAY
STOP PLAYBACK...
STOP

Cordova - Plugin Test - stop audio playback

Image - Cordova app - Plugin Test - stop audio playback 2


Android Emulator - Nexus_5X_API_24_x8...
8:32
PLUGIN TEST - AUDIO
PLUGIN MEDIA PLAYER
CHOOSE A MEDIA FILE TO PLAYBACK...
PLAY
OK

Alert

stop playing audio...& release!

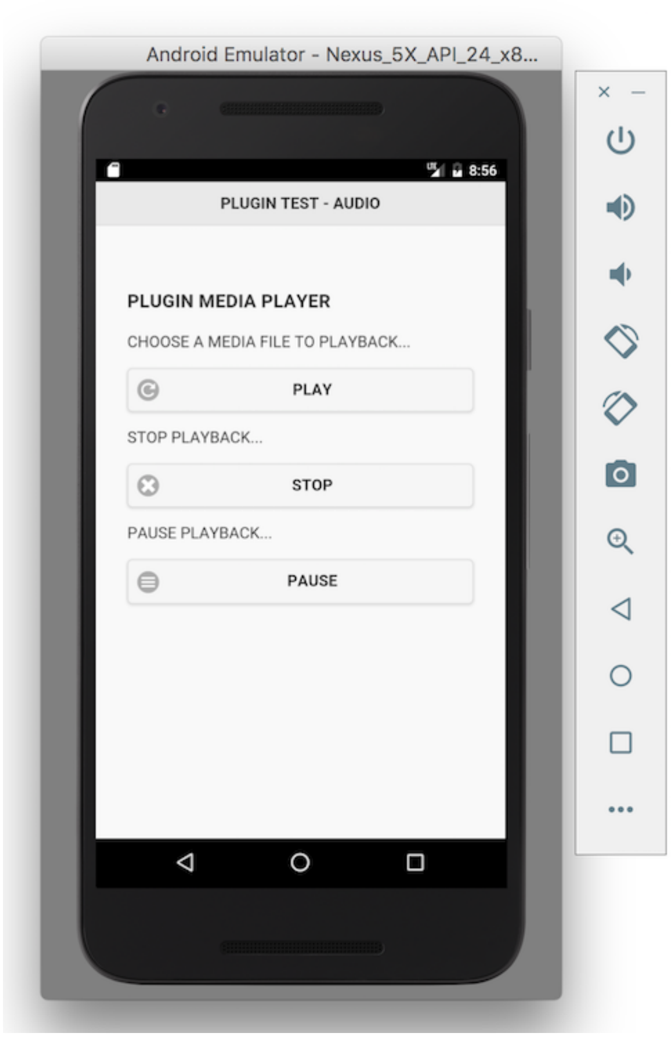
Cordova - Plugin Test - stop audio playback 2

Cordova app - working with plugins - pause button - part 1

- follow similar pattern to add initial pause button to app's HTML
- then add our custom `pauseAudio()` method
  - *handles pausing of current media object*

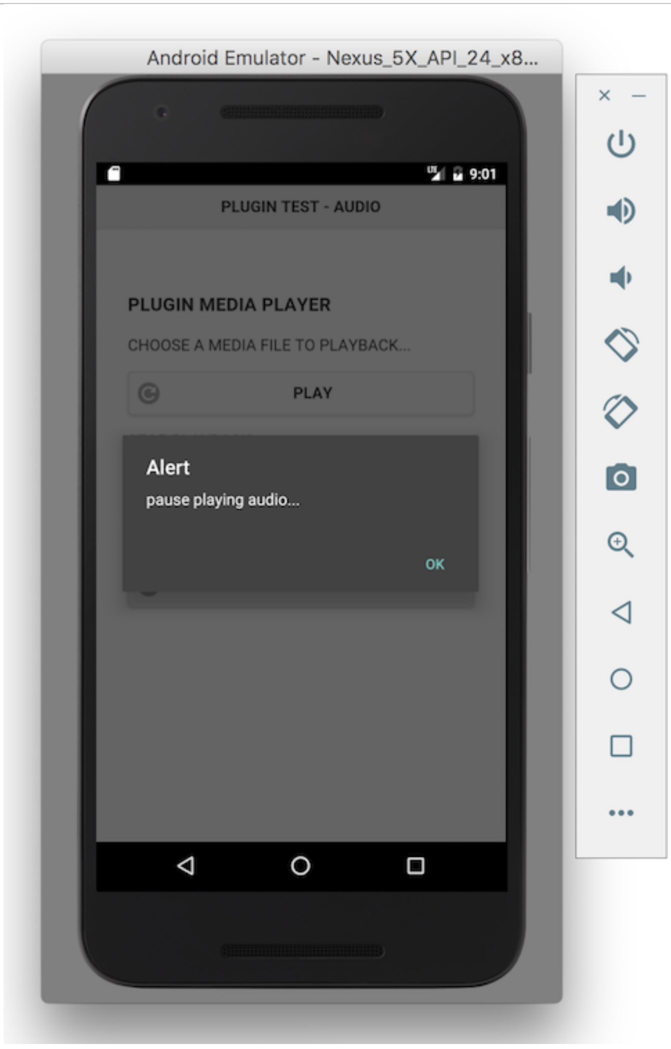
```
//pause audio file
function pauseAudio() {
    //pause audio playback
  $audio.pause();
}
```

Image - Cordova app - Plugin Test - pause audio playback



Cordova - Plugin Test - pause audio playback

Image - Cordova app - Plugin Test - pause audio playback 2

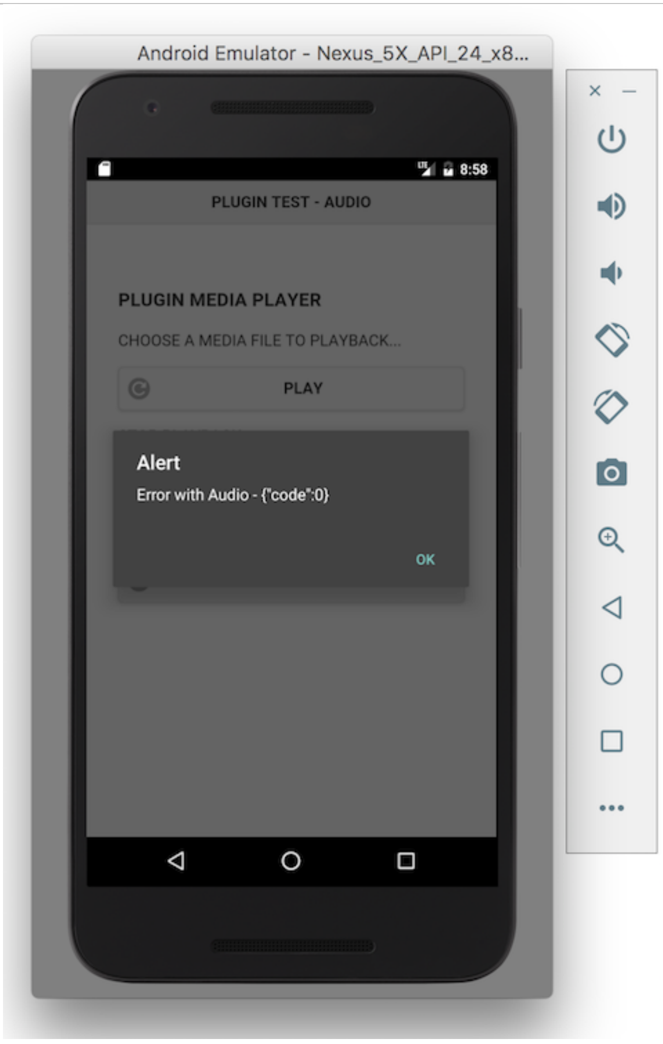


Cordova - Plugin Test - pause audio playback 2

Cordova app - working with plugins - pause button - part 2

- this logic works but it introduces issues and errors, e.g.
  - *start playback of audio and then pause*
  - *then touch play again*
  - *audio will restart from the start of the audio file*
  - *not ideal user experience...*
- an error will be thrown, e.g.
  - *press pause once, then twice...*
  - *error will be thrown for the call to the* `pause()` *method*

Image - Cordova app - Plugin Test - pause audio playback 3


Android Emulator - Nexus_5X_API_24_x8...
8:58
PLUGIN TEST - AUDIO
PLUGIN MEDIA PLAYER
CHOOSE A MEDIA FILE TO PLAYBACK...
PLAY
OK

Alert

Error with Audio - {"code":0}

Cordova - Plugin Test - pause audio playback 3

References

- Carmody, Tim., *Fighting Words: Defining "Mobile" and "Computer"* Wired. 11.08.2010. http://www.wired.com/2010/11/fighting-words-defining-mobile-and-computer/
- Cordova Doc
  - *deviceready*
  - *Events*
  - *File plugin*
  - *Media plugin*
- Google Developers - Progressive Web Apps