Cordova - Guide - Working with Plugins

- Dr Nick Hayward

A brief overview and introduction to Apache Cordova plugins, and their general usage in application development.

Contents

Intro

We're now going to start looking at some of the plugins available for Cordova, and we'll start with the simple task of playing back some media. In this case, an audio file, which we'll include as default with the application.

What this allows us to do, quite simply, is test our initial Cordova blueprint with jQuery Mobile, add some existing plugins, and then see how they fit together to create a coherent, basic application.

Create a project

So, the first thing we need to do is create our new project, which, for the sake of elaborate naming schemes, I've chosen to call **plugintest1**

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

As we're building this test application for use with Android, obviously we'll need to add support for that platform

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

Likewise, if we want to support other platforms, such as iOS, we'll also need to add the appropriate platform support.

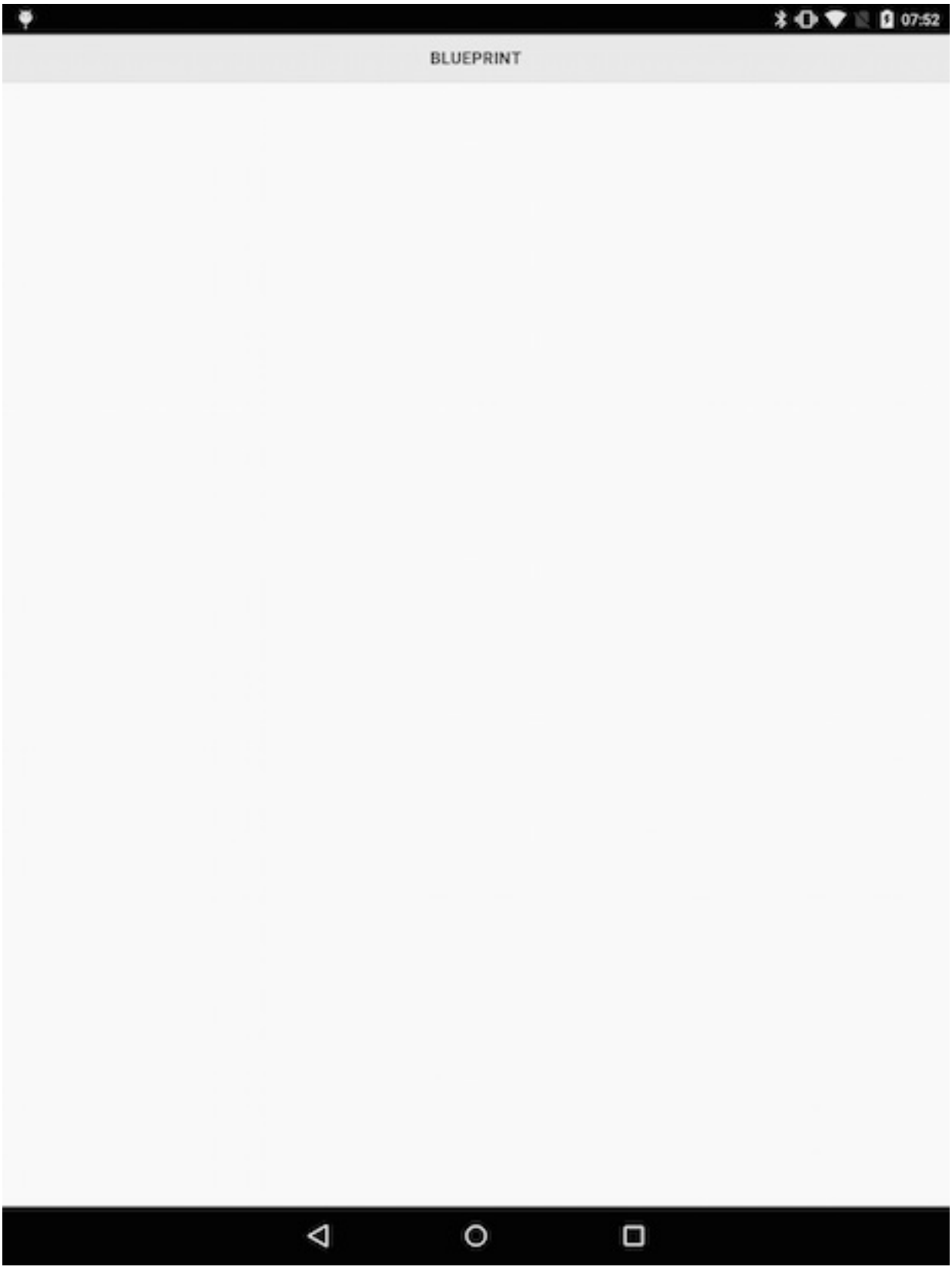
Then, we can transfer our default `www` directory, and start updating some of the settings in the `config.xml` file for the application.

For example, we'll want to update some of the metadata for the application, including author, description, and name.

Then, we can quickly run this base for our new application, just to test that we have a working starting point for the rest of our application.

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

Image - Cordova app - Plugin Test 1 - getting started



Add plugins

The next thing to do as we setup our plugin test application is to add our required plugins. For this test application, we are going to add the **device**, **file**, and **media** plugins.

The **device** plugin is added to allow us to check and read information about the current device, in effect our Android phone or tablet.

The **file** plugin is required to allow us to simply access the device's underlying filesystem.

And, the fun plugin is **media**, which helps us record and playback media files, including our required audio test file.

To add these plugins to our project, we can issue the following Cordova commands,

n.b. NPM install is now preferred option...

```
//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
```

Then, to ensure these new plugins are applied to our current project, we run the following build command in the project's working directory

```
cordova build
```

Check for any errors in the output, and hopefully we're OK to continue with development.

Update `index.html`

The next thing we need to do is update our `index.html` page to create the basic layout to allow us to load and use media files.

As before, we're going to use a single page application structure, and we'll include our content categories for `header` and `main`. Within `main`, we're going to add a content grouping, which will allow us to organise our playback buttons for the media. We can then wrap these buttons in a `div` with a data-role set to `fieldcontain`. This simply signifies that we have a contiguous group of form, input, or data elements that we can then arrange and style in a similar manner.

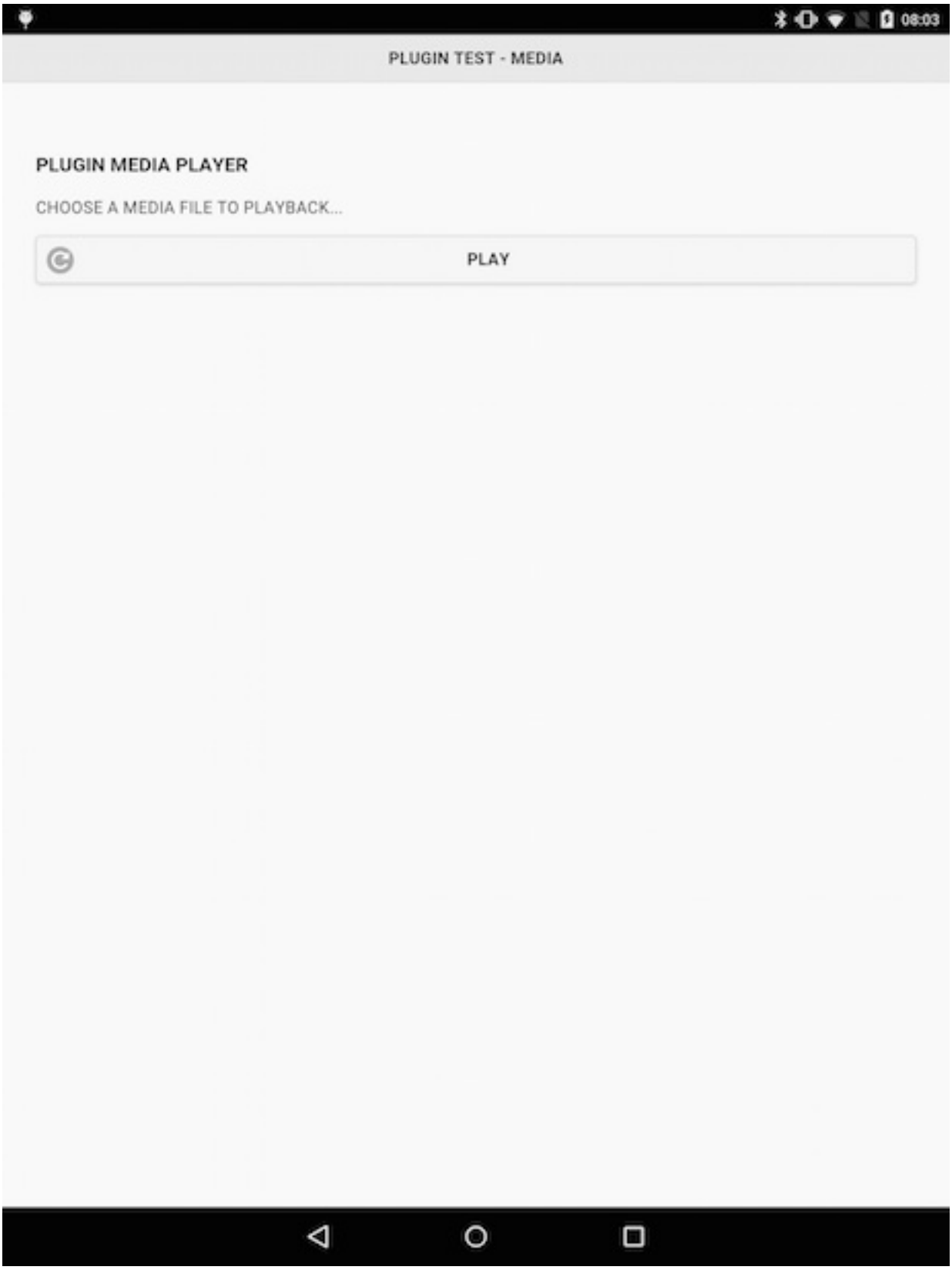
We then use this grouping to add our play button, allowing us to load our sample file using the installed plugins. For the button, we're going to use an `input` element with type set to button, a useful little icon, although not essential, a default value for the button itself set to **Play**.

`index.html` page structure

```
<!-- homepage -->
<div data-role="page" id="home">
        <div data-role="header">
    <h3>plugin test - media</h3>
        </div><!-- /header -->
  <div role="main" class="ui-content">
    <!-- container for media options... -->
    <div data-role="content">
      <!-- group buttons &c. -->
      <div data-role="fieldcontain">
        <h3>Plugin Media Player</h3>
        <p>choose a media file to playback...</p>
        <input type="button" id="playAudio" data-icon="refresh" value="Play" />
      </div>
    </div>
        </div><!-- /content -->
```

```
</div><!-- /homepage -->
```

Image - Cordova app - Plugin Test 1 - getting started



Add some logic

Let's now add some logic to our application, starting with some updates to our JavaScript to allow us to handle events. For example, we'll need to add handlers for listeners for each button we add to the application, including the initial **play** button.

We'll add this code to our application's custom JavaScript file.

So, the first thing we need to do is setup the application in response to Cordova's `deviceready` event.

Quick reminder - this is the event that informs us that the installed plugins are now loaded and ready for use. There's no point trying to load our handlers before the applicable plugins are available in the system.

We'll need to add a function for the `deviceready` event, which will allow us to bind our handler for the tap listener on the **play** button

```
functon onDeviceReady() {
  $("#playAudio").on("tap", function(e) {
    //add code for action...
  });
}
```

`onDeviceReady()`

We can also add any other required, initial functions later to this same start-up function. We then wrap this initial function in our main application loader, which checks that the device is ready, and then adds any required handlers &c..

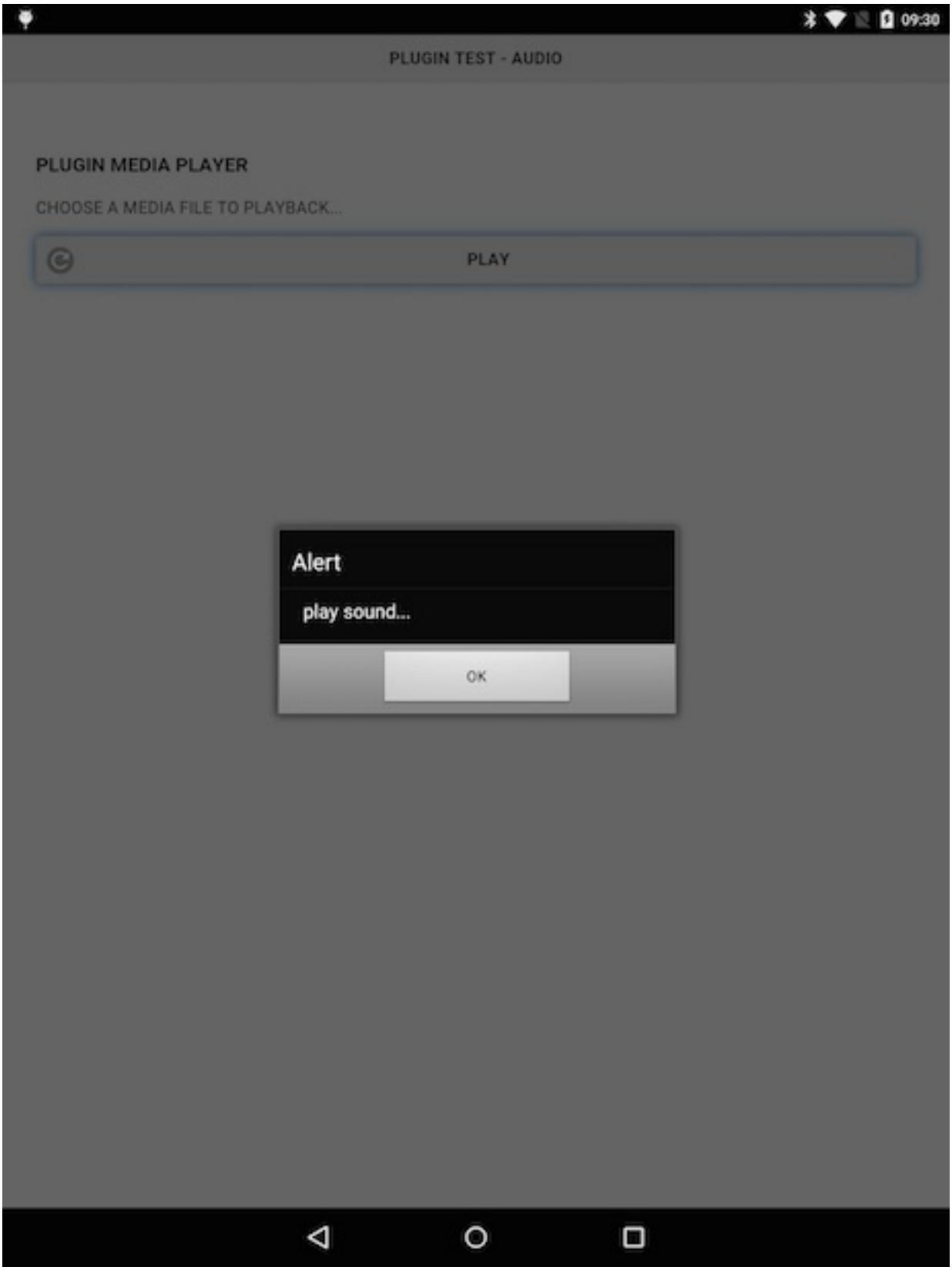
```
(function() {
        //check for page initialisation and #home
        $(document).on("pageinit", "#home", function(e) {
                //Cancels the event if it is cancelable, without stopping further
propagation of the event
                e.preventDefault();

                //loader function after deviceready event returns
                function onDeviceReady() {
                        //play audio
                        $("#playAudio").on("tap", function(e) {
                                //audio playback logic
                                alert("play sound...");
                        });

                }
                //as deviceready returns load onDeviceReady()
                $(document).on("deviceready", onDeviceReady);
        });

})();
```

Image - Cordova app - Plugin Test 1 - getting started



Audio playback logic

We've now setup and tested the logic of a basic app, and added our handlers for `deviceready` and clicking the audio playback button.

However, we still can't playback an audio file. So, we need to return to our logic for the `#playAudio` button. A simple way to playback audio, thereby testing the media plugin is as follows,

```
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!");
}
```

and we can add associated media loaders, for the audio file, and basic error checks in case the media file is missing, corrupt &c..

```
function errorReport(error) {
  alert("Error with Audio - " + JSON.stringify(error));
}

function buildURL(file) {
  if (device.platform.toLowerCase() === "android") {
    var $androidFile = "/android_asset/www/" + file;
    return $androidFile;
  }
}
```

Image - Cordova app - Plugin Test 1 - getting started

Update media playback

As we saw with the plugin test for media playback, we can allow a user to play a default audio file in their app. By simply touching a UI element, in this example a button, the app requested a file from the host file system, which was then loaded for playback within the Cordova app.

By leveraging native functionality through the included plugins for **device**, **file**, and **media** we went from the user interaction in the UI, through the JS API for the plugins, down to the native device itself.

However, the selected file would continue to play in the app unless a user refreshed the audio stream by once more touching the playback button, or simply exited the app to stop all playback. Not an ideal solution for a media

application.

An obvious addition to any media playback app is the simple option to pause, and then stop, a media stream whilst the app is running.

So, we need to add options within the UI for pause and stop functionality.

Add a **stop** button

To add functionality to **stop** a currently playing media file, we need to consider how the audio, for example, will be stopped.

For example, do we allow a user to explicitly stop the audio stream by pressing a button, or perhaps we simply playback a short snippet of the audio stream and then automatically stop playback. Your choices will be informed by the requirements of the app itself, and the options you wish to offer your users. You might not want a user to be able to stop audio playback, at least for a given period of time.

For example, you might have some introductory music or theme for a game, which needs to play for at least the first few seconds as the loading screen is shown.

The basic logic, however, regardless of choice offered to the user, is inherently the same. We are using the available methods exposed by the Cordova Media plugin, which we call in our app's JS logic.

These initial methods include,

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

We'll go through each one, and see how we can use them to update the functionality of our app.

stop button

We can start to update our existing app by adding a **stop** button to the UI, which will allow our user to simply tap a button to stop playback of the current media file.

```
<p>Stop playback...</p>
<input type="button" id="stopAudio" data-icon="delete" value="Stop" />
```

We can then update our app's JS logic to listen for a tap event on the stop button, and then call the stop method on the **media** object.

```
//button - stop audio
$("#stopAudio").on("tap", function(e) {
        //stop audio logic
        e.preventDefault();
  //call custom method to handle stopping audio...
        stopAudio();
});
```

We can then add the logic for our custom method to stop the audio, which we 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!");
```

```
}
```

However, whilst the logic itself is fine, it still won't stop the audio playing. The issue is the variable `$audio`, which currently has a restricted local scope to the `playAudio()` method. For now, a simple solution is to alter the scope of the initial property for `$audio` itself, which we can now set in the initial `onDeviceReady()` method.

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

We can update the value of this variable in the `playAudio()` method, and then stop it in the `stopAudio()` method. We also need to call the `release()` method on the current media object to ensure that system resources are freed up. This is particularly important for Android devices.

Image - Cordova app - Plugin Test - stop audio playback

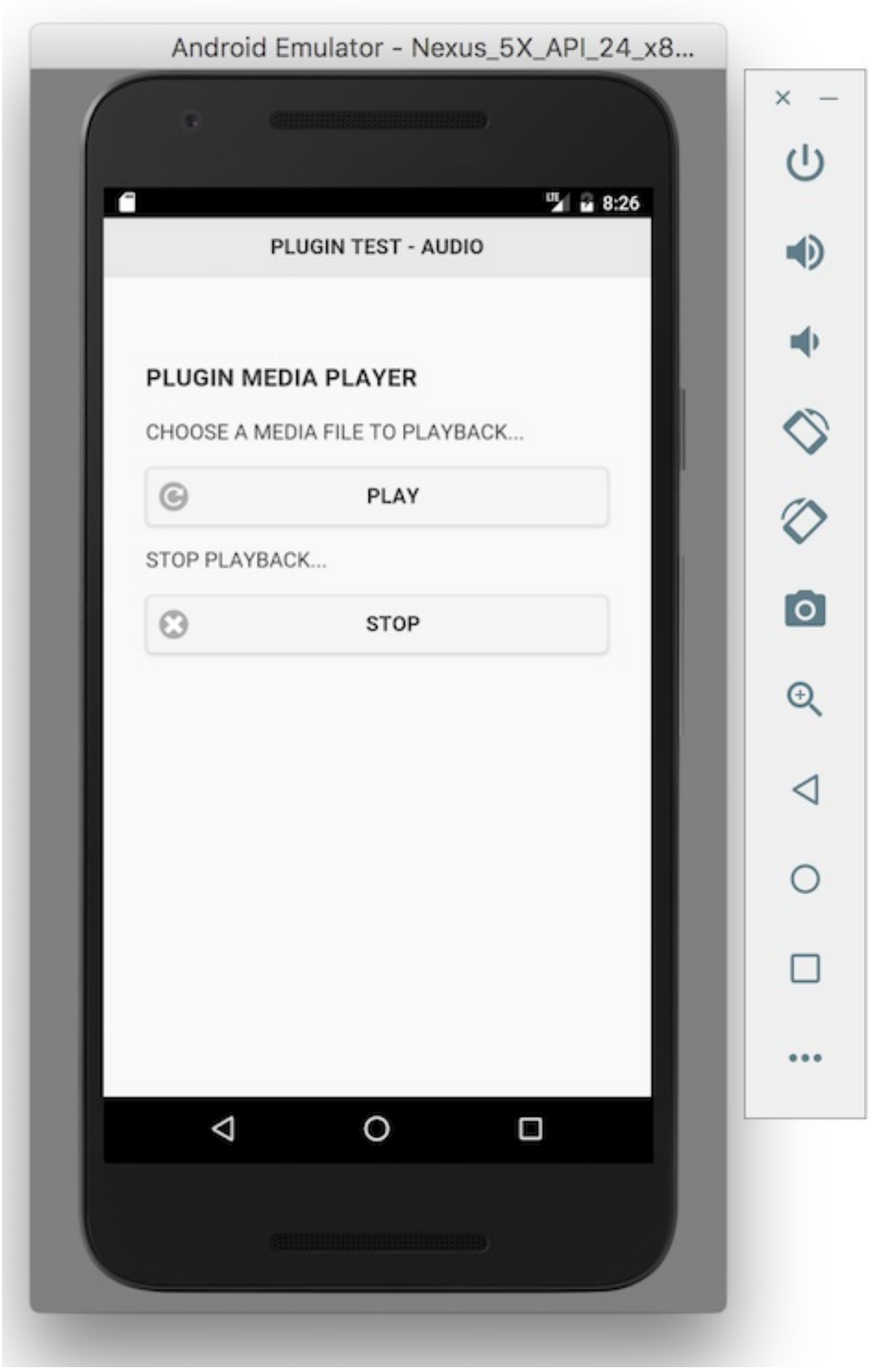
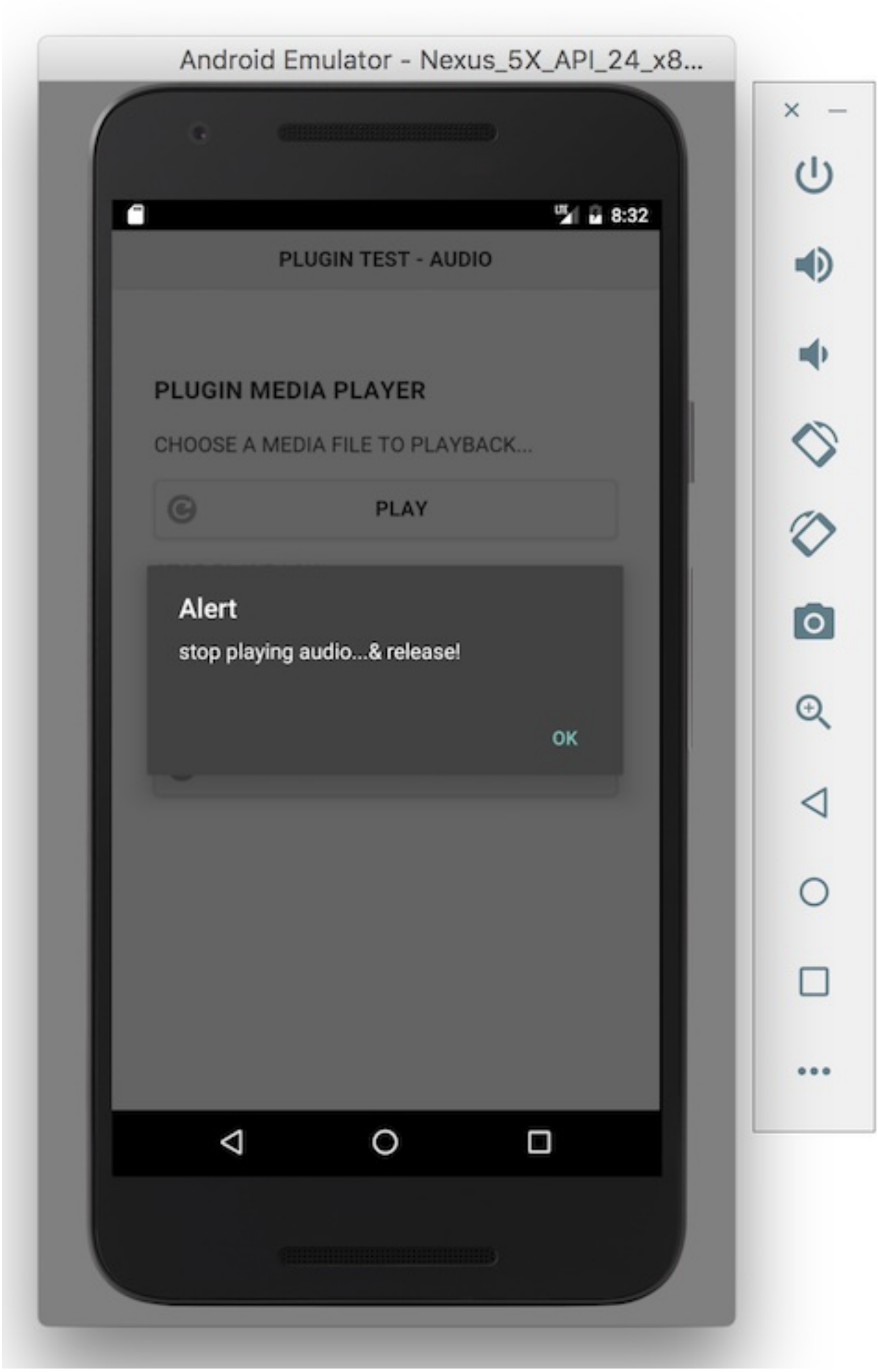


Image - Cordova app - Plugin Test - stop audio playback 2



Add a **pause** button

We can follow a similar pattern to add our initial pause button to the app's HTML,

```
<p>Pause playback...</p>
<input type="button" id="pauseAudio" data-icon="bars" value="Pause" />
```

Then, we can add a basic listener for the tap event on the pause button,

```
//button - pause audio
$("#pauseAudio").on("tap", function(e) {
        //pause audio logic
        e.preventDefault();
        //call custom method to handle pausing audio...
        pauseAudio();
});
```

and then add our custom `pauseAudio()` method to handle the pausing of the current media object.

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

Image - Cordova app - Plugin Test - pause audio playback

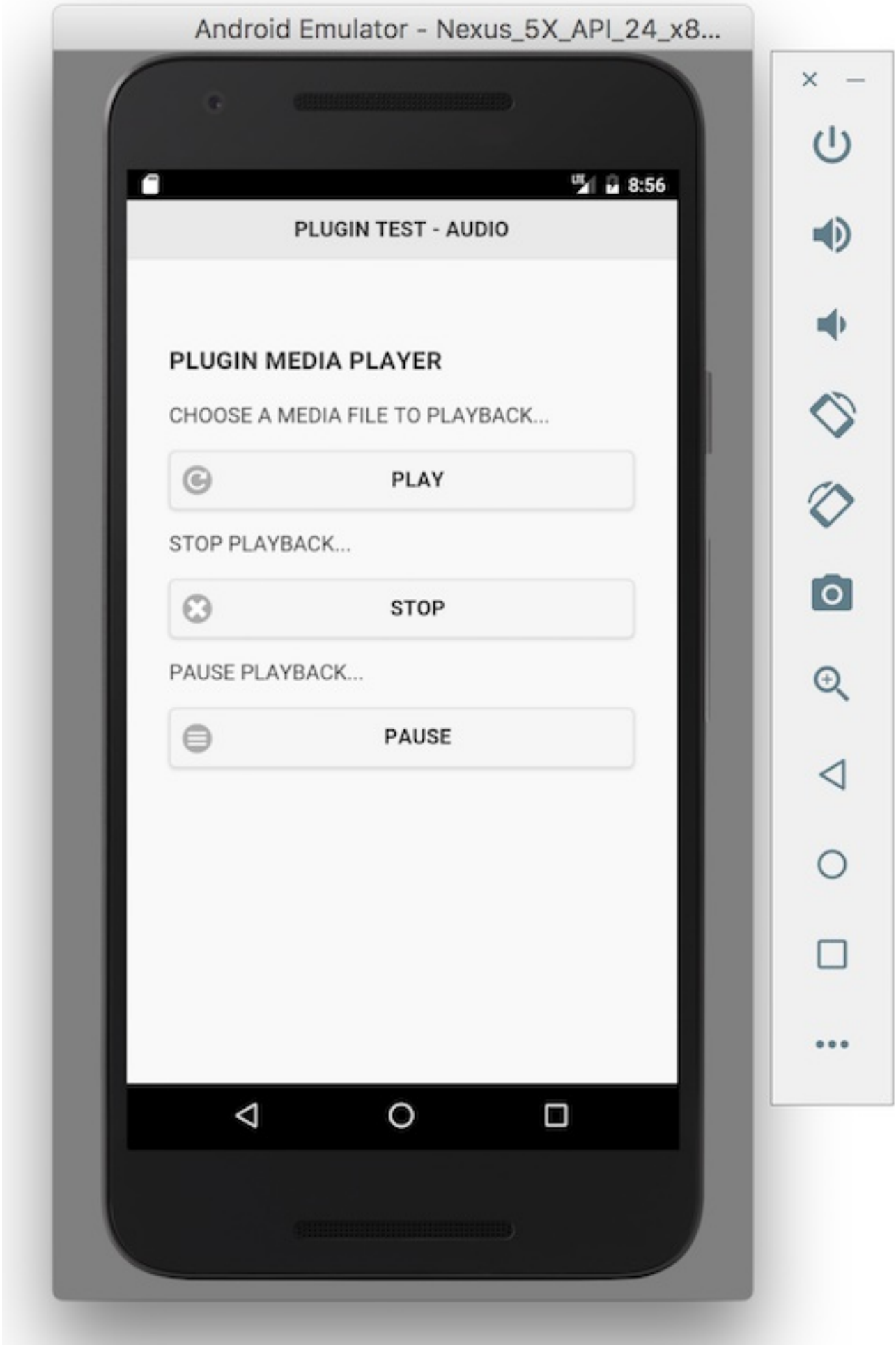
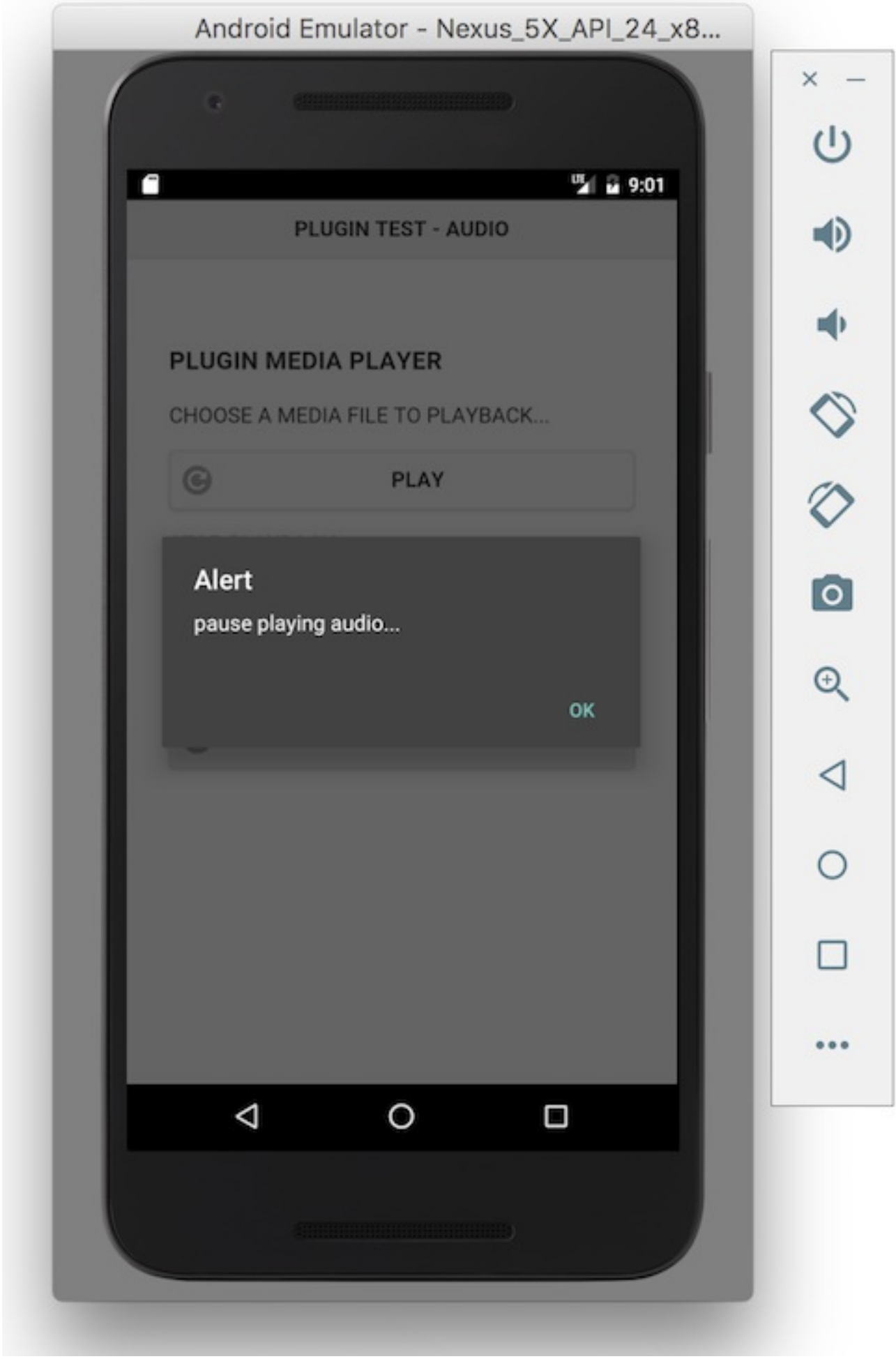


Image - Cordova app - Plugin Test - pause audio playback 2

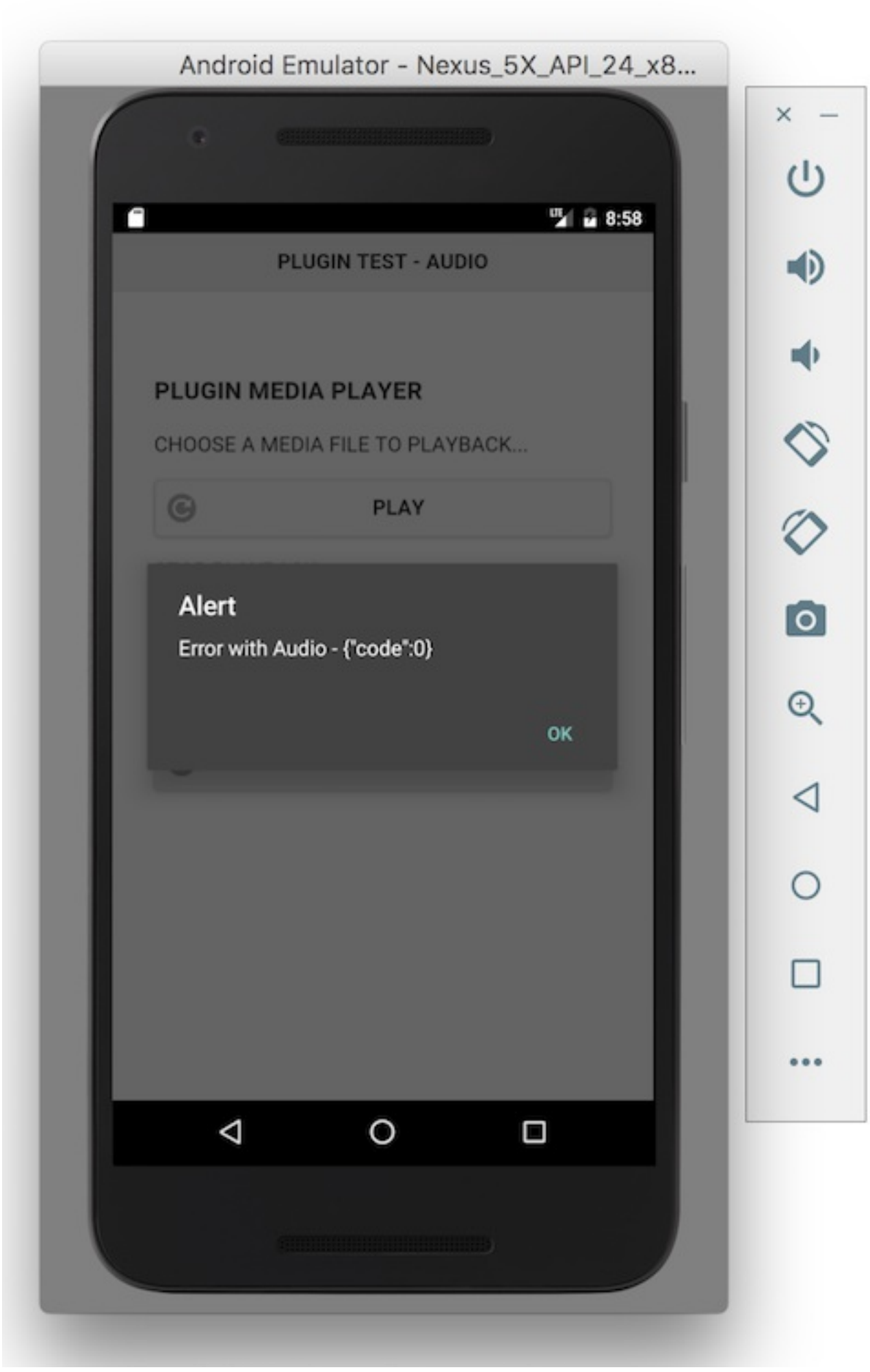


Whilst this works, and will pause the audio playback, it will quickly introduce errors into the app's logic. For example,

- start playback of audio and then pause. Then touch play again, and the audio will restart from the start of the audio file. Not an ideal solution for pause.
- press pause once, then twice, and an error will be thrown for the call to the `pause()` method.

So, we need to monitor the state of the audio track, and ensure that both play and pause is able to update and monitor this state.

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



For the current iteration of this app, we can monitor this change in the playback with a simple property we attach to the

scope for the `onDeviceReady()` method.

This property will then be available to the play, pause, and stop methods within the app.

So, the first thing we'll do is set our new property,

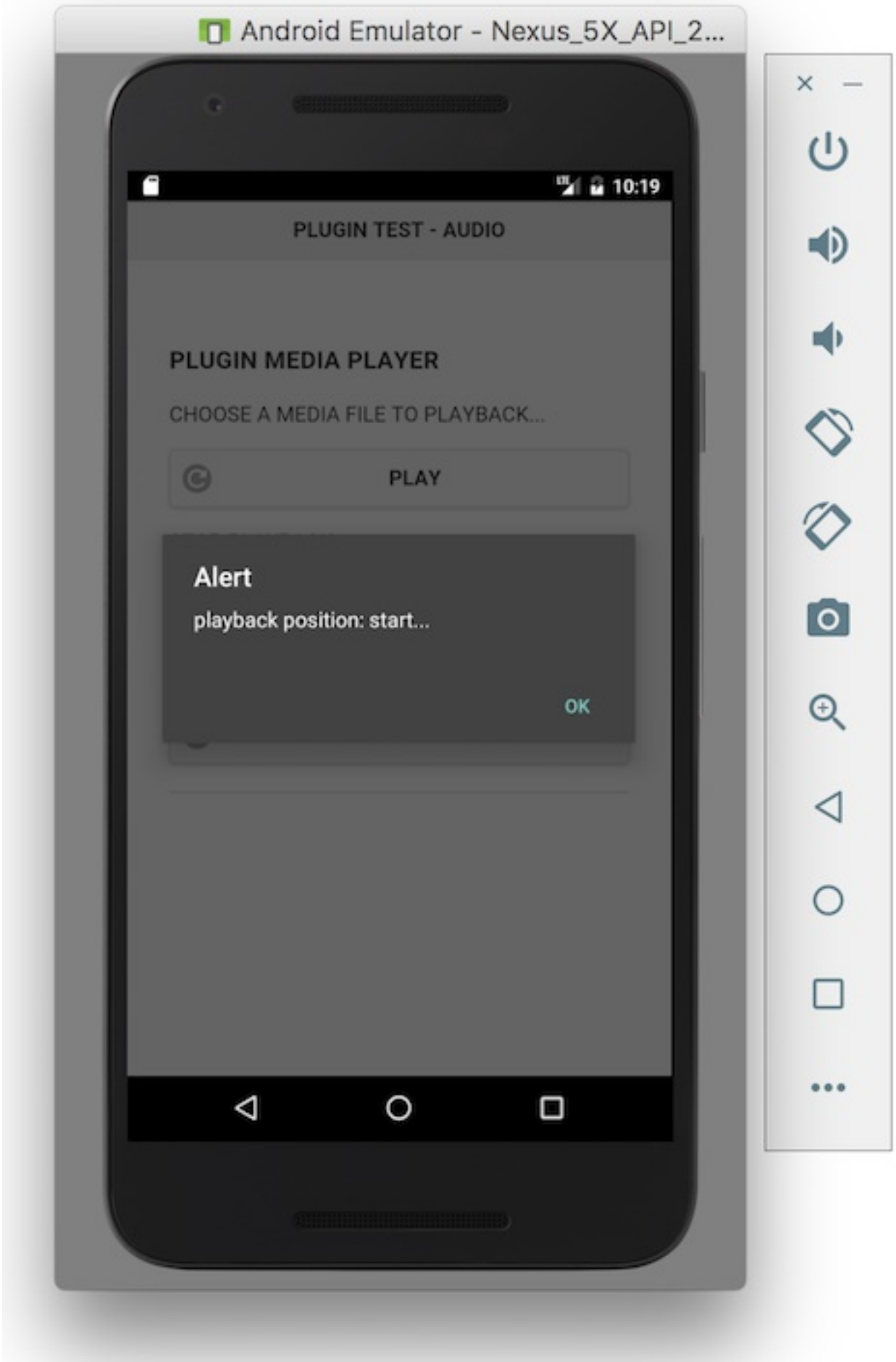
```
function onDeviceReady() {
                //set initial properties
                var $audio;
                var $audioPosn = 0;
...
}
```

We now have two properties we can monitor and update. The variable `$audioPosn` has been set to a default value of 0, which we can check as we start to playback an audio file,

```
//check current audio position
if ($audioPosn > 1) {
  $audio.play();
  alert("playback position: " + $audioPosn + " secs");
} else {
        $audio.play();
        alert("playback position: start...");
}
```

We can also use this property to output the current playback position, reset for cancelling, and so on.

Image - Cordova app - Plugin Test - update playback 1

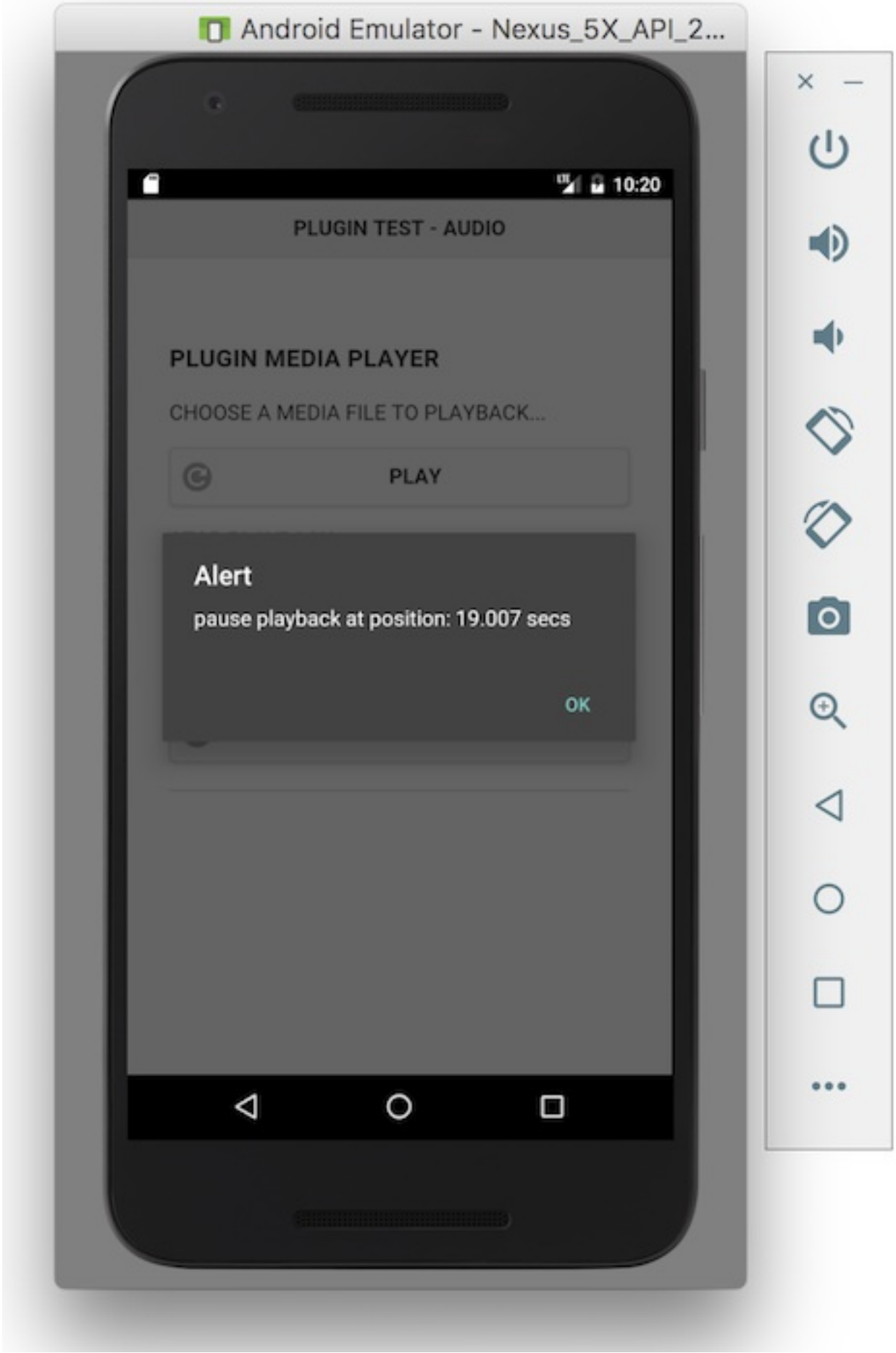


If we now pause a playing audio stream, we need to be able to get the current playback position for the audio file, and then update our `$audioPosn` property. We can do this in the `pauseAudio()` method using the `getCurrentPosition()` method, which is available on the `media` object itself.

```
$audio.getCurrentPosition(
  // success callback
  function (position) {
    if (position > -1) {
                        $audioPosn = position;
      alert("pause playback at position: " + position + " secs");
    }
  }, // error callback
     function (e) {
      ...
    }
);
```

In the success callback for this method, `position` will return a value in seconds. We can use this to set the current value for the property `$audioPosn`, which we can use elsewhere in the app.

Image - Cordova app - Plugin Test - update playback 2



Now that we can successfully pause our audio playback, and store the current value for the pause position in the audio stream, we need to update our audio playback.

First, we need check the current position in the audio stream,

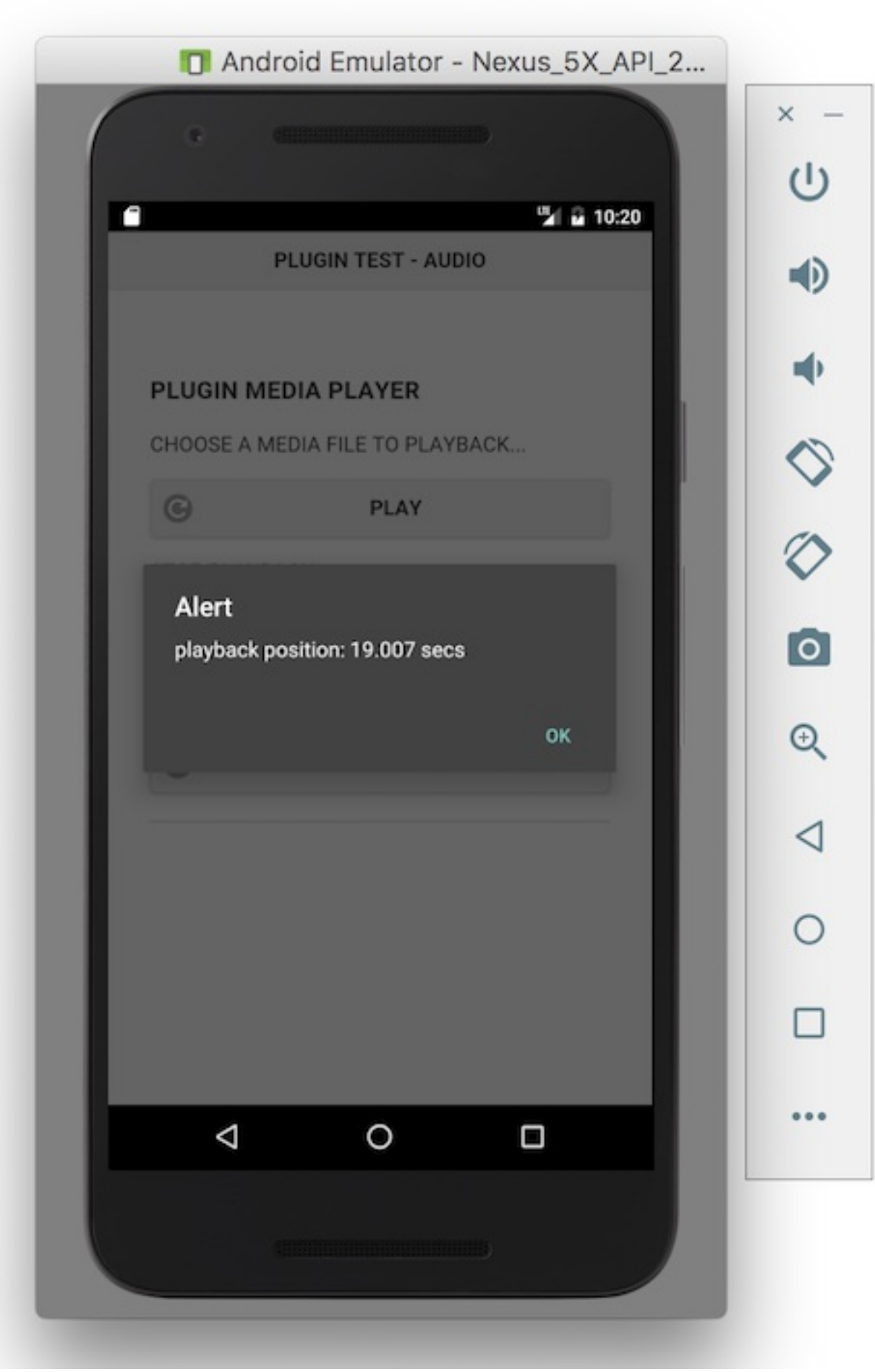
```
//check current audio position
if ($audioPosn > 1) {
        $audio.seekTo($audioPosn*1000);
        $audio.play();
        alert("playback position: " + $audioPosn + " secs");
} else {
        $audio.play();
        alert("playback position: start...");
}
```

This is why we updated the `playAudio()` method to check the value of the `$audioPosn` property. We can now use this value to seek to the current position in the audio stream, using the `seekTo()` method exposed by the media object.

However, because this method expects a time in milliseconds we'll need to update the value for our `$audioPosn` property.

Now, when we press the play button after pausing the audio stream it will restart at the correct position.

Image - Cordova app - Plugin Test - update playback 3



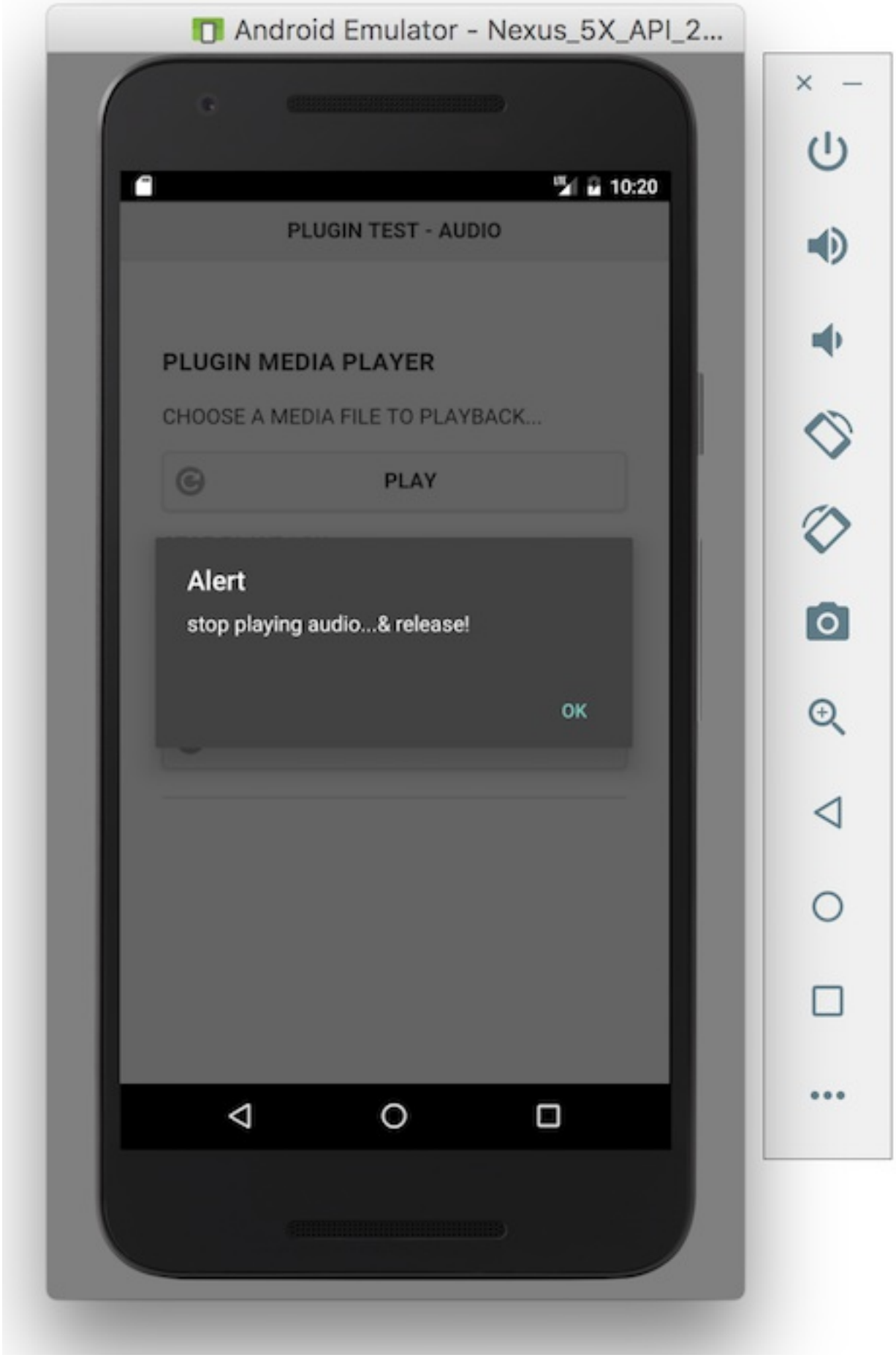
Update **stop** button

As a final touch for now, at least with the buttons, we need to update the logic for our **stop** button so it resets the value of the `$audioPosn` property or the audio stream will always restart at the previous pause value.

```
//stop audio file
function stopAudio() {
  //stop audio playback
        $audio.stop();
        //reset $audioPosn
        $audioPosn = 0;
        //release audio - important for android resources...
        $audio.release();
        //just for testing
        alert("stop playing audio...& release!");
}
```

So, we can simply reset the value of the `$audioPosn` property to `0` for now. Then, when the `playAudio()` method checks this property it will start the audio stream at the start.

Image - Cordova app - Plugin Test - update playback 4



Check **current playback position**

We've now seen how we can check the current position of a playing audio file.

There are many different options for outputting this value, including appending its value to an element in the DOM, showing a dialogue, and so on.

Again, how we use the value of this property is up to us as developers, and will naturally be informed by the requirements of the app. It may only be necessary to use this value internally, to help with the app's logic, or we may need to output this result to the user.

Further considerations for plugin usage

A few updates and modifications for a media app

- update logic for app
  - checks for event order, property values, &c.
- indicate playback has started
  - **without** alerts...
- update state of buttons in response to app state
  - highlights, colour updates...
- inactive buttons and controls when not needed
  - update state of buttons...
- grouping of buttons to represent media player
  - add correct icons, playback options...
- metadata for audio file
  - title, artist, length of track...
- image for track playing
  - thumbnail for track, album...
- track description
- notification for track playing
- persist track data and choice in cache for reload...
- ...