Cordova - Overview

Cordova is a platform for building hybrid mobile applications using HTML, CSS and JavaScript.

The official documentation gives us the definition of the Cordova –

"Apache Cordova is an open-source mobile development framework. It allows you to use standard web technologies such as HTML5, CSS3, and JavaScript for crossplatform development, avoiding each mobile platform native development language. Applications execute within wrappers targeted to each platform, and rely on standards-compliant API bindings to access each device's sensors, data, and network status."

Cordova Features

Let us now understand the features of Cordova in brief.

**Command Line Interface (Cordova CLI)**

This tool can be used for starting projects, building processes for different platforms, installing plugins and lot of other useful things that make the development process easier. You will learn how to use the Command Line Interface in the subsequent chapters.

**Cordova Core Components**

Cordova offers a set of core components that every mobile application needs. These components will be used for creating base of the app so we can spend more time to implement our own logic.

**Cordova Plugins**

Cordova offers API that will be used for implementing native mobile functions to our JavaScript app.
License

Cordova is licensed under the Apache License, Version 2.0. Apache and the Apache feather logos are trademarks of The Apache Software Foundation.

Cordova Advantages

We will now discuss the advantages of Cordova.

- Cordova offers one platform for building hybrid mobile apps so we can develop one app that will be used on different mobile platforms – IOS, Android, Windows Phone, Amazon-fireos, blackberry, Firefox OS, Ubuntu and tizien.
- It is faster to develop hybrid app then native app so Cordova can save on the development time.
- Since we are using JavaScript when working with Cordova, we don't need to learn platform specific programming languages.
- There are many community add-ons that can be used with Cordova, these have several libraries and frameworks, which are optimized for working with it.

Cordova Limitations

Following are the limitations of Cordova.

- Hybrid apps are slower than native ones so it is not optimal to use Cordova for large apps that require lots of data and functionality.
- Cross browser compatibility can create lots of issues. Most of the time we are building apps for different platforms so the testing and optimizing can be time consuming since we need to cover large number of devices and operating systems.
- Some plugins have compatibility issues with different devices and platforms. There are also some native APIs that are not yet supported by Cordova.

Cordova - Environment Setup

In this chapter, we will understand the Environment Setup of Cordova. To begin with the setup, we need to first install a few components. The components are listed in the following table.

<table>
<thead>
<tr>
<th>S.No</th>
<th>Software &amp; Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NodeJS and NPM</td>
</tr>
</tbody>
</table>
NodeJS is the platform needed for Cordova development. Check out our NodeJS Environment Setup for more details.

<table>
<thead>
<tr>
<th>2</th>
<th>Android SDK</th>
</tr>
</thead>
<tbody>
<tr>
<td>For Android platform, you need to have Android SDK installed on your machine. Check out Android Environment Setup for more details.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3</th>
<th>XCode</th>
</tr>
</thead>
<tbody>
<tr>
<td>For iOS platform, you need to have xCode installed on your machine. Check out iOS Environment Setup for more details</td>
<td></td>
</tr>
</tbody>
</table>

Installing Cordova

Before we start, you need to know that we will use Windows command prompt in our tutorial.

**Step 1 - Installing git**

Even if you don’t use git, it should be installed since Cordova is using it for some background processes. You can download git here. After you install git, open your environment variable.

- Right-Click on Computer
- Properties
- Advanced System settings
- Environment Variables
- System Variables
- Edit

Copy the following at the end of the variable value field. This is default path of the git installation. If you installed it on a different path you should use that instead of our example code below.

```
;C:\Program Files (x86)\Git\bin;C:\Program Files (x86)\Git\cmd
```

Now you can type `git` in your command prompt to test if the installation is successful.

**Step 2 - Installing Cordova**

This step will download and install Cordova module globally. Open the command prompt and run the following –
C:\Users\username>npm install -g cordova

You can check the installed version by running –

C:\Users\username>cordova -v

This is everything you need to start developing the Cordova apps on Windows operating system. In our next tutorial, we will show you how to create first application.

**Cordova - First Application**

We have understood how to install Cordova and set up the environment for it. Once everything is ready, we can create our first hybrid Cordova application.

**Step 1 - Creating App**

Open the directory where you want the app to be installed in command prompt. We will create it on desktop.

C:\Users\username\Desktop>cordova create CordovaProject io.cordova.hellocordova CordovaApp

- **CordovaProject** is the directory name where the app is created.
- **io.cordova.hellocordova** is the default reverse domain value. You should use your own domain value if possible.
- **CordovaApp** is the title of your app.

**Step 2 - Adding Platforms**

You need to open your project directory in the command prompt. In our example, it is the **CordovaProject**. You should only choose platforms that you need. To be able to use the specified platform, you need to have installed the specific platform SDK. Since we are developing on windows, we can use the following platforms. We have already installed Android SDK, so we will only install android platform for this tutorial.

C:\Users\username\Desktop\CordovaProject>cordova platform add android

There are other platforms that can be used on Windows OS.

C:\Users\username\Desktop\CordovaProject>cordova platform add wp8
C:\Users\username\Desktop\CordovaProject>cordova platform add amazon-fireos
C:\Users\username\Desktop\CordovaProject>cordova platform add windows
C:\Users\username\Desktop\CordovaProject>cordova platform add blackberry10
C:\Users\username\Desktop\CordovaProject>cordova platform add firefoxos

If you are developing on Mac, you can use –

$ cordova platform add iOS
$ cordova platform add amazon-fireos

THANKS FOR READING – VISIT OUR WEBSITE: www.educatererindia.com
$ cordova platform add android
$ cordova platform add blackberry10
$ cordova platform add firefoxos

You can also remove platform from your project by using −
C:\Users\username\Desktop\CordovaProject>cordova platform rm android

Step 3 - Building and Running

In this step we will build the app for a specified platform so we can run it on mobile device or emulator.

C:\Users\username\Desktop\CordovaProject>cordova build android

Now we can run our app. If you are using the default emulator you should use −
C:\Users\username\Desktop\CordovaProject>cordova emulate android

If you want to use the external emulator or real device you should use −
C:\Users\username\Desktop\CordovaProject>cordova run android

NOTE − We will use the **Genymotion android emulator** since it is faster and more responsive than the default one. You can find the emulator [here](#). You can also use real device for testing by enabling **USB debugging** from the options and connecting it to your computer via USB cable. For some devices, you will also need to install the USB driver.

Once we run the app, it will install it on the platform we specified. If everything is finished without errors, the output should show the default start screen of the app.
In our next tutorial, we will show you how to configure the Cordova Application.

**Cordova - config.xml File**

The `config.xml` file is the place where we can change the configuration of the app. When we created our app in the last tutorial, we set reverse domain and name. The values can be changed in the `config.xml` file. When we create the app, the default config file will also be created.
The following table explains configuration elements in `config.xml`.

### `config.xml` Configuration Table

<table>
<thead>
<tr>
<th>S.No</th>
<th>Element &amp; Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>widget</code></td>
</tr>
<tr>
<td></td>
<td>The app reverse domain value that we specified when creating the app.</td>
</tr>
<tr>
<td>2</td>
<td><code>name</code></td>
</tr>
<tr>
<td></td>
<td>The name of the app that we specified when creating the app.</td>
</tr>
<tr>
<td>3</td>
<td><code>description</code></td>
</tr>
<tr>
<td></td>
<td>Description for the app.</td>
</tr>
<tr>
<td>4</td>
<td><code>author</code></td>
</tr>
<tr>
<td></td>
<td>Author of the app.</td>
</tr>
<tr>
<td>5</td>
<td><code>content</code></td>
</tr>
<tr>
<td></td>
<td>The app's starting page. It is placed inside the <code>www</code> directory.</td>
</tr>
<tr>
<td>6</td>
<td><code>plugin</code></td>
</tr>
<tr>
<td></td>
<td>The plugins that are currently installed.</td>
</tr>
<tr>
<td>7</td>
<td><code>access</code></td>
</tr>
</tbody>
</table>
Used to control access to external domains. The default **origin** value is set to *
which means that access is allowed to any domain. This value will not allow
some specific URLs to be opened to protect information.

### 8 allow-intent

Allows specific URLs to ask the app to open. For example, `<allow-intent href
="tel:*" />` will allow tel: links to open the dialer.

### 9 platform

The platforms for building the app.

## Cordova - Storage

We can use storage API available for storing data on the client apps. This will help
the usage of the app when the user is offline and it can also improve performance. Since this tutorial is for beginners, we will show you how to use **local storage**. In one of our later tutorials, we will show you the other plugins that can be used.

### Step 1 - Adding Buttons

We will create four buttons in the **index.html** file. The buttons will be located inside the **div class = "app"** element.

```html
<button id = "setLocalStorage">SET LOCAL STORAGE</button>
<button id = "showLocalStorage">SHOW LOCAL STORAGE</button>
<button id = "removeProjectFromLocalStorage">REMOVE PROJECT</button>
<button id = "getLocalStorageByKey">GET BY KEY</button>
```

It will produce the following screen –
Step 2 - Adding Event Listeners

Cordova security policy doesn’t allow inline events so we will add event listeners inside index.js files. We will also assign `window.localStorage` to a `localStorage` variable that we will use later.

```javascript
document.getElementById("setLocalStorage").addEventListener("click", setLocalStorage);

document.getElementById("showLocalStorage").addEventListener("click", showLocalStorage);

document.getElementById("removeProjectFromLocalStorage").addEventListener("click", removeProjectFromLocalStorage);

(document.getElementById("getLocalStorageByKey")).addEventListener("click", getLocalStorageByKey);
```
var localStorage = window.localStorage;

Step 3 - Creating Functions

Now we need to create functions that will be called when the buttons are tapped. First function is used for adding data to local storage.

```javascript
function setLocalStorage() {
    localStorage.setItem("Name", "John");
    localStorage.setItem("Job", "Developer");
    localStorage.setItem("Project", "Cordova Project");
}
```

The next one will log the data we added to console.

```javascript
function showLocalStorage() {
    console.log(localStorage.getItem("Name"));
    console.log(localStorage.getItem("Job"));
    console.log(localStorage.getItem("Project"));
}
```

If we tap **SET LOCAL STORAGE** button, we will set three items to local storage. If we tap **SHOW LOCAL STORAGE** afterwards, the console will log items that we want.

```
John  index.js:70
Developer  index.js:71
Cordova Project  index.js:72
```

Let us now create function that will delete the project from local storage.

```javascript
function removeProjectFromLocalStorage() {
    localStorage.removeItem("Project");
}
```

If we click the **SHOW LOCAL STORAGE** button after we deleted the project, the output will show **null** value for the project field.

```
null  index.js:72
```

THANKS FOR READING – VISIT OUR WEBSITE: www.educatererindia.com
We can also get the local storage elements by using the `key()` method which will take the index as an argument and return the element with corresponding index value.

```javascript
function getLocalStorageByKey() {
    console.log(localStorage.key(0));
}
```

Now when we tap the **GET BY KEY** button, the following output will be displayed.

```
Job  index.js:83
```

**NOTE**

When we use the `key()` method, the console will log the **job** instead of the **name** even though we passed argument 0 to retrieve the first object. This is because the local storage is storing data in alphabetical order.

The following table shows all the available local storage methods.

<table>
<thead>
<tr>
<th>S.No</th>
<th>Methods &amp; Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>setItem(key, value)</code></td>
</tr>
<tr>
<td></td>
<td>Used for setting the item to local storage.</td>
</tr>
<tr>
<td>2</td>
<td><code>getItem(key)</code></td>
</tr>
<tr>
<td></td>
<td>Used for getting the item from local storage.</td>
</tr>
<tr>
<td>3</td>
<td><code>removeItem(key)</code></td>
</tr>
<tr>
<td></td>
<td>Used for removing the item from local storage.</td>
</tr>
<tr>
<td>4</td>
<td><code>key(index)</code></td>
</tr>
<tr>
<td></td>
<td>Used for getting the item by using the <strong>index</strong> of the item in local storage. This helps sort the items alphabetically.</td>
</tr>
<tr>
<td>5</td>
<td><code>length()</code></td>
</tr>
<tr>
<td></td>
<td>Used for retrieving the number of items that exists in local storage.</td>
</tr>
</tbody>
</table>
clear()
Used for removing all the key/value pairs from local storage.

## Cordova - Events

There are various events that can be used in Cordova projects. The following table shows the available events.

<table>
<thead>
<tr>
<th>S.No</th>
<th>Events &amp; Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>deviceReady</strong></td>
</tr>
<tr>
<td></td>
<td>This event is triggered once Cordova is fully loaded. This helps to ensure that no Cordova functions are called before everything is loaded.</td>
</tr>
<tr>
<td>2</td>
<td><strong>pause</strong></td>
</tr>
<tr>
<td></td>
<td>This event is triggered when the app is put into background.</td>
</tr>
<tr>
<td>3</td>
<td><strong>resume</strong></td>
</tr>
<tr>
<td></td>
<td>This event is triggered when the app is returned from background.</td>
</tr>
<tr>
<td>4</td>
<td><strong>backbutton</strong></td>
</tr>
<tr>
<td></td>
<td>This event is triggered when the back button is pressed.</td>
</tr>
<tr>
<td>5</td>
<td><strong>menubutton</strong></td>
</tr>
<tr>
<td></td>
<td>This event is triggered when the menu button is pressed.</td>
</tr>
<tr>
<td>6</td>
<td><strong>searchbutton</strong></td>
</tr>
<tr>
<td></td>
<td>This event is triggered when the Android search button is pressed.</td>
</tr>
<tr>
<td>7</td>
<td><strong>startcallbutton</strong></td>
</tr>
<tr>
<td></td>
<td>This event is triggered when the start call button is pressed.</td>
</tr>
<tr>
<td>8</td>
<td><strong>endcallbutton</strong></td>
</tr>
</tbody>
</table>

---

THANKS FOR READING – VISIT OUR WEBSITE [www.educatererindia.com](http://www.educatererindia.com)
This event is triggered when the end call button is pressed.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td><strong>volumedownbutton</strong>&lt;br&gt;This event is triggered when the volume down button is pressed.</td>
</tr>
<tr>
<td>10</td>
<td><strong>volumeupbutton</strong>&lt;br&gt;This event is triggered when the volume up button is pressed.</td>
</tr>
</tbody>
</table>

### Using Events

All of the events are used almost the same way. We should always add event listeners in our js instead of the inline event calling since the **Cordova Content Security Policy** doesn’t allow inline Javascript. If we try to call event inline, the following error will be displayed.

![Error message](image)

The right way of working with events is by using `addEventListener`. We will understand how to use the `volumeupbutton` event through an example.

```javascript
function callbackFunction() {
  alert('Volume Up Button is pressed!');
}
```

Once we press the **volume up** button, the screen will display the following alert.
Handling Back Button

We should use the Android back button for app functionalities like returning to the previous screen. To implement your own functionality, we should first disable the back button that is used to exit the App.

```javascript
document.addEventListener("backbutton", onKeyDown, false);

function onKeyDown(e) {
    e.preventDefault();
    alert('Back Button is Pressed!');
}
```
Now when we press the native Android back button, the alert will appear on the screen instead of exiting the app. This is done by using the `e.preventDefault()` command.

Cordova - Back Button

Handling Back Button

You will usually want to use Android back button for some app functionality like returning to previous screen. To be able to implement your own functionality, you first need to disable exiting the app when the back button is pressed.

```javascript
document.addEventListener("backbutton", onBackKeyDown, false);

function onBackKeyDown(e) {
```

THANKS FOR READING – VISIT OUR WEBSITE: www.educatererindia.com
Now when we press the native Android back button, the alert will appear on the screen instead of exiting the app. This is done by using `e.preventDefault()`.  

```javascript
    e.preventDefault();
    alert('Back Button is Pressed!');
```  

Cordova - Plugman  

Cordova Plugman is a useful command line tool for installing and managing plugins. You should use `plugman` if your app needs to run on one specific platform. If you want to create a cross-platform app you should use `cordova-cli` which will modify plugins for different platforms.
Step 1 - Installing Plugman

Open the **command prompt** window and run the following code snippet to install plugman.

```
C:\Users\username\Desktop\CordovaProject>npm install -g plugman
```

Step 2 - Installing Plugins

To understand how to install the Cordova plugin using plugman, we will use the Camera plugin as an example.

```
C:\Users\username\Desktop\CordovaProject>plugman install --platform android --project platforms\android --plugin cordova-plugin-camera
```

We need to consider three parameters as shown above.

- **--platform** – platform that we are using (android, ios, amazon-fireos, wp8, blackberry10).
- **--project** – path where the project is built. In our case, it is **platforms\android** directory.
- **--plugin** – the plugin that we want to install.

If you set valid parameters, the command prompt window should display the following output.

```
Installing "cordova-plugin-camera" for android
```

Additional Methods

You can use the **uninstall** method in similar way.

```
C:\Users\username\Desktop\CordovaProject>plugman uninstall --platform android --project platforms\android --plugin cordova-plugin-camera
```

The **command prompt** console will display the following output.

```
Uninstalling cordova-plugin-camera from android
Removing "cordova-plugin-camera"
```

Plugman offers some additional methods that can be used. The methods are listed in the following table.

<table>
<thead>
<tr>
<th>S.No</th>
<th>Method &amp; Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>install</strong></td>
</tr>
<tr>
<td></td>
<td>Used for installing Cordova plugins.</td>
</tr>
<tr>
<td></td>
<td>Command</td>
</tr>
<tr>
<td>---</td>
<td>---------</td>
</tr>
<tr>
<td>2</td>
<td><code>uninstall</code></td>
</tr>
<tr>
<td>3</td>
<td><code>fetch</code></td>
</tr>
<tr>
<td>4</td>
<td><code>prepare</code></td>
</tr>
<tr>
<td>5</td>
<td><code>adduser</code></td>
</tr>
<tr>
<td>6</td>
<td><code>publish</code></td>
</tr>
<tr>
<td>7</td>
<td><code>unpublish</code></td>
</tr>
<tr>
<td>8</td>
<td><code>search</code></td>
</tr>
<tr>
<td>9</td>
<td><code>config</code></td>
</tr>
<tr>
<td>10</td>
<td><code>create</code></td>
</tr>
<tr>
<td>11</td>
<td><code>platform</code></td>
</tr>
</tbody>
</table>

**Additional Commands**

If you are stuck, you can always use the `plugman help` command. The version can be checked by using `plugman -v`. To search for the plugin, you can use `plugman`.

THANKS FOR READING – VISIT OUR WEBSITE: www.educatererindia.com
search and finally you can change the plugin registry by using the plugman config set registry command.

NOTE
Since Cordova is used for cross platform development, in our subsequent chapters we will use Cordova CLI instead of Plugman for installing plugins.

Cordova - Battery Status
This Cordova plugin is used for monitoring device's battery status. The plugin will monitor every change that happens to device’s battery.

Step 1 - Installing Battery Plugin
To install this plugin, we need to open the command prompt window and run the following code.

```
C:\Users\username\Desktop\CordovaProject>cordova plugin add cordova-plugin-battery-status
```

Step 2 - Add Event Listener
When you open the index.js file, you will find the onDeviceReady function. This is where the event listener should be added.

```
window.addEventListener("batterystatus", onBatteryStatus, false);
```

Step 3 - Create Callback Function
We will create the onBatteryStatus callback function at the bottom of the index.js file.

```
function onBatteryStatus(info) {
    alert("BATTERY STATUS: Level: " + info.level + " isPlugged: " + info.isPlugged);
}
```

When we run the app, an alert will be triggered. At the moment, the battery is 100% charged.

When the status is changed, a new alert will be displayed. The battery status shows that the battery is now charged 99%.
If we plug in the device to the charger, the new alert will show that the `isPlugged` value is changed to `true`. 
Additional Events

This plugin offers two additional events besides the `batterystatus` event. These events can be used in the same way as the `batterystatus` event.

<table>
<thead>
<tr>
<th>S.No</th>
<th>Event &amp; Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>batterylow</code></td>
</tr>
<tr>
<td></td>
<td>The event is triggered when the battery charge percentage reaches low value. This value varies with different devices.</td>
</tr>
</tbody>
</table>
The event is triggered when the battery charge percentage reaches critical value. This value varies with different devices.

Cordova - Camera

This plugin is used for taking photos or using files from the image gallery.

Step 1 - Install Camera Plugin

Run the following code in the command prompt window to install this plugin.

```bash
C:\Users\username\Desktop\CordovaProject>cordova plugin add cordova-plugin-camera
```

Step 2 - Adding Button and Image

Now, we will create the button for calling the camera and img where the image will be displayed once taken. This will be added to index.html inside the div class = "app" element.

```html
<button id="cameraTakePicture">TAKE PICTURE</button>
<img id="myImage"/>
```

Step 3 - Adding Event Listener

The event listener is added inside the onDeviceReady function to ensure that Cordova is loaded before we start using it.

```javascript
document.getElementById("cameraTakePicture").addEventListener("click", cameraTakePicture);
```

Step 4 - Adding Functions (taking photo)

We will create the cameraTakePicture function that is passed as a callback to our event listener. It will be fired when the button is tapped. Inside this function, we will call the navigator.camera global object provided by the plugin API. If taking picture is successful, the data will be sent to the onSuccess callback function, if not, the alert with error message will be shown. We will place this code at the bottom of index.js.

```javascript
function cameraTakePicture() {
    navigator.camera.getPicture(onSuccess, onFail, {
        quality: 50,
```

THANKS FOR READING – VISIT OUR WEBSITE www.educatererindia.com
When we run the app and press the button, native camera will be triggered.
When we take and save picture, it will be displayed on screen.
The same procedure can be used for getting image from the local file system. The only difference is the function created in the last step. You can see that the `sourceType` optional parameter has been added.

**Step 1**

```
C:\Users\username\Desktop\CordovaProject>cordova plugin add cordova-plugin-camera
```

**Step 2**

```
<button id="cameraGetPicture">GET PICTURE</button>
```

**Step 3**

```
document.getElementById("cameraGetPicture").addEventListener("click", cameraGetPicture);
```
Function `cameraGetPicture()`:

```javascript
function cameraGetPicture() {
    navigator.camera.getPicture(onSuccess, onFail, { quality: 50,
        destinationType: Camera.DestinationType.DATA_URL,
        sourceType: Camera.PictureSourceType.PHOTOLIBRARY
    });
}
```

Function `onSuccess(imageURL)`:

```javascript
function onSuccess(imageURL) {
    var image = document.getElementById('myImage');
    image.src = imageURL;
}
```

Function `onFail(message)`:

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

When we press the second button, the file system will open instead of the camera so we can choose the image that is to be displayed.
This plugin offers lots of optional parameters for customization.

<table>
<thead>
<tr>
<th>S.No</th>
<th>Parameter &amp; Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>quality</td>
</tr>
<tr>
<td></td>
<td>Quality of the image in the range of 0-100. Default is 50.</td>
</tr>
<tr>
<td>2</td>
<td>destinationType</td>
</tr>
<tr>
<td></td>
<td>DATA_URL or 0 Returns base64 encoded string.</td>
</tr>
<tr>
<td></td>
<td>FILE_URI or 1 Returns image file URI.</td>
</tr>
<tr>
<td><strong>NATIVE_URI</strong> or 2</td>
<td>Returns image native URI.</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td><strong>sourceType</strong></td>
<td></td>
</tr>
<tr>
<td>PHOTOLIBRARY or 0</td>
<td>Opens photo library.</td>
</tr>
<tr>
<td>CAMERA or 1</td>
<td>Opens native camera.</td>
</tr>
<tr>
<td>SAVEDPHOTOALBUM or 2</td>
<td>Opens saved photo album.</td>
</tr>
<tr>
<td><strong>allowEdit</strong></td>
<td>Allows image editing.</td>
</tr>
<tr>
<td><strong>encodingType</strong></td>
<td></td>
</tr>
<tr>
<td>JPEG or 0</td>
<td>Returns JPEG encoded image.</td>
</tr>
<tr>
<td>PNG or 1</td>
<td>Returns PNG encoded image.</td>
</tr>
<tr>
<td><strong>targetWidth</strong></td>
<td>Image scaling width in pixels.</td>
</tr>
<tr>
<td><strong>targetHeight</strong></td>
<td>Image scaling height in pixels.</td>
</tr>
<tr>
<td><strong>mediaType</strong></td>
<td></td>
</tr>
<tr>
<td>PICTURE or 0</td>
<td>Allows only picture selection.</td>
</tr>
<tr>
<td>VIDEO or 1</td>
<td>Allows only video selection.</td>
</tr>
<tr>
<td>ALLMEDIA or 2</td>
<td>Allows all media type selection.</td>
</tr>
<tr>
<td><strong>correctOrientation</strong></td>
<td>Used for correcting orientation of the image.</td>
</tr>
<tr>
<td><strong>saveToPhotoAlbum</strong></td>
<td>Used to save the image to the photo album.</td>
</tr>
</tbody>
</table>
**Cordova - Contacts**

This plugin is used for accessing the contacts database of the device. In this tutorial we will show you how to create, query and delete contacts.

**Step 1 - Install Contacts Plugin**

```
C:\Users\username\Desktop\CordovaProject>cordova plugin add cordova-plugin-contacts
```

**Step 2 - Adding Buttons**

The button will be used for calling the `createContact` function. We will place it in the `div class = "app"` in `index.html` file.

```html
<button id = "createContact">ADD CONTACT</button>
<button id = "findContact">FIND CONTACT</button>
<button id = "deleteContact">DELETE CONTACT</button>
```

**Step 2 - Add Event Listeners**

Open `index.js` and copy the following code snippet into the `onDeviceReady` function.

```javascript
document.getElementById("createContact").addEventListener("click", createContact);
document.getElementById("findContact").addEventListener("click", findContact);
document.getElementById("deleteContact").addEventListener("click", deleteContact);
```

**Step 3A - Callback Function**

`(navigator.contacts.create)`

Now, we do not have any contacts stored on the device.
Our first callback function will call the `navigator.contacts.create` method where we can specify the new contact data. This will create a contact and assign it to the `myContact` variable but it will not be stored on the device. To store it, we need to call the `save` method and create success and error callback functions.

```javascript
function createContact() {
    var myContact = navigator.contacts.create({
        "displayName": "Test User"
    });
    myContact.save(contactSuccess, contactError);

    function contactSuccess() {
        alert("Contact is saved!");
    }
    function contactError() {
        alert("Error saving contact.");
    }
}
```
When we click the **ADD CONTACT** button, new contact will be stored to the device contact list.

```javascript
function contactError(message) {
    alert('Failed because: ' + message);
}
```
Step 3B - Callback Function (navigator.contacts.find)

Our second callback function will query all contacts. We will use the `navigator.contacts.find` method. The options object has filter parameter which is used to specify the search filter. `multiple = true` is used since we want to return all contacts from device. The `field` key to search contacts by `displayName` since we used it when saving contact.

After the options are set, we are using `find` method to query contacts. The alert message will be triggered for every contact that is found.

```javascript
function findContacts() {
    var options = new ContactFindOptions();
    options.filter = "";
    options.multiple = true;
    fields = ["displayName"];  
navigator.contacts.find(fields, contactfindSuccess, contactfindError, options);

    function contactfindSuccess(contacts) {
        for (var i = 0; i < contacts.length; i++) {
            alert("Display Name = " + contacts[i].displayName);
        }
    }

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

When we press the FIND CONTACT button, one alert popup will be triggered since we have saved only one contact.
Step 3C - Callback function (delete)

In this step, we will use the find method again but this time we will set different options. The `options.filter` is set to search that `Test User` which has to be deleted. After the `contactfindSuccess` callback function has returned the contact we want, we will delete it by using the `remove` method that requires its own success and error callbacks.

```javascript
function deleteContact() {
    var options = new ContactFindOptions();
    options.filter = "Test User";
    options.multiple = false;
}
```
fields = ['displayName'];
navigator.contacts.find(fields, contactfindSuccess, contactfindError, options);

function contactfindSuccess(contacts) {
    var contact = contacts[0];
    contact.remove(contactRemoveSuccess, contactRemoveError);
}

function contactRemoveSuccess(contact) {
    alert("Contact Deleted");
}

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

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

Now, we have only one contact stored on the device. We will manually add one more to show you the deleting process.
We will now click the **DELETE CONTACT** button to delete the **Test User**. If we check the contact list again, we will see that the **Test User** does not exist anymore.
Cordova - Device

This plugin is used for getting information about the user’s device.

**Step 1 - Installing Device Plugin**

To install this plugin, we need to run the following snippet in the command prompt.

```
C:\Users\username\Desktop\CordovaProject>cordova plugin add cordova-plugindevice
```

**Step 2 - Adding Button**
We will be using this plugin the same way we used the other Cordova plugins. Let us add a button in the `index.html` file. This button will be used for getting information about the device.

```html
<button id="cordovaDevice">CORDOVA DEVICE</button>
```

**Step 3 - Adding Event Listener**

Cordova plugins are available after the `deviceready` event so we will place the event listener inside the `onDeviceReady` function in `index.js`.

```javascript
document.getElementById("cordovaDevice").addEventListener("click", cordovaDevice);
```

**Step 4 - Creating Function**

The following function will show how to use all possibilities the plugin provides. We will place it in `index.js`.

```javascript
function cordovaDevice() {
    alert("Cordova version: " + device.cordova + "\n"
            "Device model: " + device.model + "\n"
            "Device platform: " + device.platform + "\n"
            "Device UUID: " + device.uuid + "\n"
            "Device version: " + device.version);
}
```

When we click the CORDOVA DEVICE button, the alert will display the Cordova version, device model, platform, UUID and device version.
Cordova - Accelerometer

The Accelerometer plugin is also called the **device-motion**. It is used to track device motion in three dimensions.

**Step 1 - Install Accelerometer Plugin**

We will install this plugin by using **cordova-CLI**. Type the following code in the **command prompt** window.

C:\Users\username\Desktop\CordovaProject>cordova plugin add cordova-plugin-device-motion

**Step 2 - Add Buttons**
In this step, we will add two buttons in the `index.html` file. One will be used for getting the current acceleration and the other will watch for the acceleration changes.

```html
<button id="getAcceleration">GET ACCELERATION</button>
<button id="watchAcceleration">WATCH ACCELERATION</button>
```

**Step 3 - Add Event Listeners**

Let us now add event listeners for our buttons to `onDeviceReady` function inside `index.js`.

```javascript
document.getElementById("getAcceleration").addEventListener("click", getAcceleration);
document.getElementById("watchAcceleration").addEventListener("click", watchAcceleration);
```

**Step 4 - Creating Functions**

Now, we will create two functions. The first function will be used to get the current acceleration and the second function will watch the acceleration and the information about the acceleration will be triggered every three seconds. We will also add the `clearWatch` function wrapped by the `setTimeout` function to stop watching acceleration after the specified time frame. The `frequency` parameter is used to trigger the callback function every three seconds.

```javascript
function getAcceleration() {
    navigator.accelerometer.getCurrentAcceleration(
        accelerometerSuccess, accelerometerError);

    function accelerometerSuccess(acceleration) {
        alert('Acceleration X: ' + acceleration.x + '
             ' + acceleration.y + '
             ' + acceleration.z + '
             ' + 'Timestamp: ' + acceleration.timestamp + 'n');
    }

    function accelerometerError() {
    }
```
Now if we press the **GET ACCELERATION** button, we will get the current acceleration value. If we press the **WATCH ACCELERATION** button, the alert will be triggered every three seconds. After third alert is shown the **clearWatch** function.
will be called and we will not get any more alerts since we set timeout to 10000 milliseconds.

Cordova - Device Orientation

Compass is used to show direction relative to geographic north cardinal point.

Step 1 - Install Device Orientation plugin

Open the command prompt window and run the following.

```
C:\Users\username\Desktop\CordovaProject>cordova plugin add cordova-plugin-device-orientation
```

Step 2 - Add Buttons
This plugin is similar to the `acceleration` plugin. Let us now create two buttons in `index.html`.

```html
<button id = "getOrientation">GET ORIENTATION</button>
<button id = "watchOrientation">WATCH ORIENTATION</button>
```

**Step 3 - Add Event Listeners**

Now, we will add `event listeners` inside the `onDeviceReady` function in `index.js`.

```javascript
document.getElementById("getOrientation").addEventListener("click", getOrientatio);
document.getElementById("watchOrientation").addEventListener("click", watchOrientation);
```

**Step 4 - Creating Functions**

We will create two functions; the first function will generate the current acceleration and the other will check on the orientation changes. You can see that we are using the `frequency` option again to keep a watch on changes that occur every three seconds.

```javascript
function getOrientatio() {
    navigator.compass.getCurrentHeading(compassSuccess, compassError);
}

function compassSuccess(heading) {
    alert('Heading: ' + heading.magneticHeading);
};

function compassError(error) {
    alert('CompassError: ' + error.code);
};

function watchOrientation(){
    var compassOptions = {
        frequency: 3000
    }
}
var watchID = navigator.compass.watchHeading(compassSuccess,
    compassError, compassOptions);

function compassSuccess(heading) {
    alert('Heading: ' + heading.magneticHeading);
    setTimeout(function() {
        navigator.compass.clearWatch(watchID);
    }, 10000);
}

function compassError(error) {
    alert('CompassError: ' + error.code);
}

Since the compass plugin is almost the same as the acceleration plugin, we will show you an error code this time. Some devices do not have the magnetic sensor that is needed for the compass to work. If your device doesn't have it, the following error will be displayed.
Cordova - Dialogs

The Cordova Dialogs plugin will call the platform native dialog UI element.

Step 1 - Installing Dialog

Type the following command in the command prompt window to install this plugin.

C:\Users\username\Desktop\CordovaProject>cordova plugin add cordova-plugin-dialogs

Step 2 - Add Buttons

Let us now open index.html and add four buttons, one for every type of dialog.

<button id = "dialogAlert">ALERT</button>
Step 3 - Add Event Listeners

Now we will add the event listeners inside the `onDeviceReady` function in `index.js`. The listeners will call the callback function once the corresponding button is clicked.

```javascript
document.getElementById("dialogAlert").addEventListener("click", dialogAlert);
document.getElementById("dialogConfirm").addEventListener("click", dialogConfirm);
document.getElementById("dialogPrompt").addEventListener("click", dialogPrompt);
document.getElementById("dialogBeep").addEventListener("click", dialogBeep);
```

Step 4A - Create Alert Function

Since we added four event listeners, we will now create the callback functions for all of them in `index.js`. The first one is `dialogAlert`.

```javascript
function dialogAlert() {
    var message = "I am Alert Dialog!";
    var title = "ALERT";
    var buttonName = "Alert Button";
    navigator.notification.alert(message, alertCallback, title, buttonName);

    function alertCallback() {
        console.log("Alert is Dismissed!");
    }
}
```

If we click the **ALERT** button, we will get see the alert dialog box.
When we click the dialog button, the following output will be displayed on the console.

```
Alert is Dismissed! index.js:76
```

**Step 4B - Create Confirm Function**

The second function we need to create is the `dialogConfirm` function.

```javascript
function dialogConfirm() {
    var message = "Am I Confirm Dialog?";
    var title = "CONFIRM";
    var buttonLabels = "YES, NO";
}
```
navigator.notification.confirm(message, confirmCallback, title, buttonLabels);

function confirmCallback(buttonIndex) {
    console.log("You clicked " + buttonIndex + " button!");
}

When the **CONFIRM** button is pressed, the new dialog will pop up.
We will click the **YES** button to answer the question. The following output will be displayed on the console.

```
You clicked 1 button! index.js:87
```

**Step 4C - Create Prompt Function**

The third function is the `dialogPrompt` function. This allows the users to type text into the dialog input element.

```javascript
function dialogPrompt() {
    var message = "Am I Prompt Dialog?";
    var title = "PROMPT";
    var buttonLabels = ['YES','NO'];
    var defaultText = "Default"

    navigator.notification.prompt(message, promptCallback, 
    title, buttonLabels, defaultText);

    function promptCallback(result) {
        console.log("You clicked " + result.buttonIndex + " button! \n" + 
        "You entered " + result.input1);
    }
}
```

The **PROMPT** button will trigger a dialog box as in the following screenshot.
In this dialog box, we have an option to type the text. We will log this text in the console, together with a button that is clicked.

Step 4D - Create Beep Function

The last one is the `dialogBeep` function. This is used for calling the audio beep notification. The `times` parameter will set the number of repeats for the beep signal.

```javascript
function dialogBeep() {
    var times = 2;
    navigator.notification.beep(times);
}
```
When we click the **BEEP** button, we will hear the notification sound twice, since the **times** value is set to **2**.

### Cordova - File System

This plugin is used for manipulating the native file system on the user’s device.

#### Step 1 - Installing File Plugin

We need to run the following code in the **command prompt** to install this plugin.

```
C:\Users\username\Desktop\CordovaProject>cordova plugin add cordova-plugin-file
```

#### Step 2 - Add Buttons

In this example, we will show you how to create file, write to file, read it and delete it. For this reason, we will create four buttons in **index.html**. We will also add **textarea** wherein, the content of our file will be shown.

```html
<button id = "createFile">CREATE FILE</button>
<button id = "writeFile">WRITE FILE</button>
<button id = "readFile">READ FILE</button>
<button id = "removeFile">DELETE FILE</button>
<textarea id = "textarea"></textarea>
```

#### Step 3 - Add Event Listeners

We will add **event listeners** in **index.js** inside the **onDeviceReady** function to ensure that everything has started before the plugin is used.

```javascript
document.getElementById("createFile").addEventListener("click", createFile);
document.getElementById("writeFile").addEventListener("click", writeFile);
document.getElementById("readFile").addEventListener("click", readFile);
document.getElementById("removeFile").addEventListener("click", removeFile);
```

#### Step 4A - Create File function

The file will be created in the apps root folder on the device. To be able to access the root folder you need to provide **superuser** access to your folders. In our case, the path to root folder is `\data\data\com.example.hello\cache`. At the moment this folder is empty.
Let us now add a function that will create the log.txt file. We will write this code in `index.js` and send a request to the file system. This method uses \texttt{WINDOW.TEMPORARY} or \texttt{WINDOW.PERSISTENT}. The size that will be required for storage is valued in bytes (5MB in our case).

```javascript
function createFile() {
    var type = window.TEMPORARY;
    var size = 5*1024*1024;

    window.requestFileSystem(type, size, successCallback, errorCallback)

    function successCallback(fs) {
```

```javascript
```
Now we can press the **CREATE FILE** button and the alert will confirm that we successfully created the file.

```javascript
fs.root.getFile('log.txt', {create: true, exclusive: true}, function(fileEntry) {
  alert('File creation successful!')
}, errorCallback);

function errorCallback(error) {
  alert("ERROR: " + error.code)
}
```
Now, we can check our apps root folder again and we can find our new file there.
Step 4B - Write File Function

In this step, we will write some text to our file. We will again send a request to the file system, and then create the file writer to be able to write *Lorem Ipsum* text that we assigned to the *blob* variable.

```javascript
function writeFile() {
    var type = window.TEMPORARY;
    var size = 5*1024*1024;
    window.requestFileSystem(type, size, successCallback, errorCallback)

    function successCallback(fs) {
        // Do something with the file system
    }
}
```
fs.root.getFile('log.txt', {create: true}, function(fileEntry) {

    fileEntry.createWriter(function(fileWriter) {

        fileWriter.onwriteend = function(e) {
            alert('Write completed.');
        };

        fileWriter.onerror = function(e) {
            alert('Write failed: ' + e.toString());
        };

        var blob = new Blob(['Lorem Ipsum'], {type: 'text/plain'});
        fileWriter.write(blob);
    }, errorCallback);
    }, errorCallback);
}

function errorCallback(error) {
    alert("ERROR: " + error.code);
}

After pressing the **WRITE FILE** button, the alert will inform us that the writing is successful as in the following screenshot.
Now we can open log.txt and see that Lorem Ipsum is written inside.
Step 4C - Read File Function

In this step, we will read the log.txt file and display it in the textarea element. We will send a request to the file system and get the file object, then we are creating reader. When the reader is loaded, we will assign the returned value to textarea.

```javascript
function readFile() {
  var type = window.TEMPORARY;
  var size = 5*1024*1024;
  window.requestFileSystem(type, size, successCallback, errorCallback)
}
function successCallback(fs) {
    fs.root.getFile('log.txt', {}, function(fileEntry) {

        fileEntry.file(function(file) {
            var reader = new FileReader();

            reader.onloadend = function(e) {
                var txtArea = document.getElementById('textarea');
                txtArea.value = this.result;
            };

            reader.readAsText(file);
        }, errorCallback);
    }, errorCallback);
}

function errorCallback(error) {
    alert("ERROR: " + error.code)
}

When we click the **READ FILE** button, the text from the file will be written inside **textarea**.
Step 4D - Delete File Function

And finally we will create function for deleting `log.txt` file.

```javascript
function removeFile() {
    var type = window.TEMPORARY;
    var size = 5*1024*1024;
    window.requestFileSystem(type, size, successCallback, errorCallback)

    function successCallback(fs) {
        fs.root.getFile('log.txt', {create: false}, function(fileEntry) {

```
We can now press the **DELETE FILE** button to remove the file from the apps root folder. The alert will notify us that the delete operation is successful.
If we check the apps root folder, we will see that it is empty.
Cordova - File Transfer

This plugin is used for uploading and downloading files.

Step 1 - Installing File Transfer Plugin

We need to open command prompt and run the following command to install the plugin.

C:\Users\username\Desktop\CordovaProject>cordova plugin add cordova-plugin-file-transfer

Step 2 - Create Buttons

In this chapter, we will show you how to upload and download files. Let’s create two buttons in index.html
Step 3 - Add Event Listeners

Event listeners will be created in `index.js` inside the `onDeviceReady` function. We are adding `click` events and `callback` functions.

```javascript
document.getElementById("uploadFile").addEventListener("click", uploadFile);
document.getElementById("downloadFile").addEventListener("click", downloadFile);
```

Step 4A - Download Function

This function will be used for downloading the files from server to device. We uploaded file to `postimage.org` to make things more simple. You will probably want to use your own server. The function is placed in `index.js` and will be triggered when the corresponding button is pressed. `uri` is the server download link and `fileURI` is the path to the DCIM folder on our device.

```javascript
function downloadFile() {
  var fileTransfer = new FileTransfer();
  var uri = encodeURI("http://s14.postimg.org/i8qvaxyup/bitcoin1.jpg");
  var fileURL = "///storage/emulated/0/DCIM/myFile";

  fileTransfer.download(
    uri, fileURL, function(entry) {
      console.log("download complete: " + entry.toURL());
    },

    function(error) {
      console.log("download error source " + error.source);
      console.log("download error target " + error.target);
      console.log("download error code" + error.code);
    },

    false,
```
Once we press the **DOWNLOAD** button, the file will be downloaded from the **postimg.org** server to our mobile device. We can check the specified folder and see that **myFile** is there.

```
headers: {
  "Authorization": "Basic dGVzdHVzZXJtYWllOnRlc3RwYXNzd29yZA=="
}
```

The console output will look like this –
Step 4B - Upload Function

Now let’s create a function that will take the file and upload it to the server. Again, we want to simplify this as much as possible, so we will use posttestserver.com online server for testing. The uri value will be linked for posting to posttestserver.

```javascript
function uploadFile() {
  var fileURL = "//storage/emulated/0/DCIM/myFile"
  var uri = encodeURI("http://posttestserver.com/post.php");
  var options = new FileUploadOptions();
  options.fileKey = "file";
  options.fileName = fileURL.substr(fileURL.lastIndexOf(\'/\')+1);
  options.mimeType = "text/plain";

  var headers = {headerParam:headerValue};
  options.headers = headers;
  var ft = new FileTransfer();
  ft.upload(fileURL, uri, onSuccess, onError, options);

  function onSuccess(r) {
    console.log("Code = "+ r.responseCode);
    console.log("Response = "+ r.response);
    console.log("Sent = " + r.bytesSent);
  }

  function onError(error) {
    alert("An error has occurred: Code = " + error.code);
    console.log("upload error source " + error.source);
    console.log("upload error target " + error.target);
  }
}
```
Now we can press the **UPLOAD** button to trigger this function. We will get a console output as confirmation that the uploading was successful.

```javascript
Code = 200
Response = Successfully dumped 0 post variables.
View it at
http://www.posttestserver.com/date/2015/12/07/08.52.0068802
5272
Post body was 0 chars long.
Sent = 147557
```

We can also check the server to make sure that the file was uploaded.

```text
HTTP_HOST = posttestserver.com
HTTP_USER_AGENT = Dalvik/1.6.0 (Linux; U; Android 4.4.4;
Samsung Galaxy S4 - 4.4.4 - API 19 - 1080x1920
Build/KTU84P)
HTTP_HEADERPARAM = headerValue
CONTENT_TYPE = multipart/form-data; boundary=+++++
UNIQUE_ID = VmXHPd8x6hIAAAARmCikAAAAN
REQUEST_TIME_FLOAT = 1449510720.4047
REQUEST_TIME = 1449510720
No Post Params.
Empty post body.

== Multipart file upload. ==
Received 1 file(s)
  0: posted name=file
      name: myFile
      type: text/plain
      error: 0
      size: 162244
Uploaded File:
http://posttestserver.com/files/2015/12/07/f_09.52.00173
7825919
```

**Cordova - Geolocation**

Geolocation is used for getting info about device's latitude and longitude.

**Step 1 - Installing Plugin**

We can install this plugin by typing the following code to **command prompt** window.

```bash
C:\Users\username\Desktop\CordovaProject>cordova plugin add cordova-plugin-geolocation
```

**Step 2 - Add Buttons**
In this tutorial we will show you how to get current position and how to watch for changes. We first need to create buttons that will call these functions.

```html
<button id = "getPosition">CURRENT POSITION</button>
<button id = "watchPosition">WATCH POSITION</button>
```

**Step 3 - Add Event Listeners**

Now we want to add event listeners when the device is ready. We will add the code sample below to `onDeviceReady` function in `index.js`.

```javascript
document.getElementById("getPosition").addEventListener("click", getPosition);
document.getElementById("watchPosition").addEventListener("click", watchPosition);
```

**Step 3 - Create Functions**

Two functions have to be created for two event listeners. One will be used for getting the current position and the other for watching the position.

```javascript
function getPosition() {
    var options = {
        enableHighAccuracy: true,
        maximumAge: 3600000
    }
    var watchID = navigator.geolocation.getCurrentPosition(onSuccess, onError, options);

    function onSuccess(position) {
        alert('Latitude: ' + position.coords.latitude + 'n' +
              'Longitude: ' + position.coords.longitude + 'n' +
              'Altitude: ' + position.coords.altitude + 'n' +
              'Accuracy: ' + position.coords.accuracy + 'n' +
              'Altitude Accuracy: ' + position.coords.altitudeAccuracy + 'n' +
              'Heading: ' + position.coords.heading + 'n' +
              'Speed: ' + position.coords.speed + 'n' +
              'Timestamp: ' + position.timestamp + 'n');
    };
```
function onError(error) {
    alert('code: ' + error.code + '\n' + 'message: ' + error.message + '\n');
}

function watchPosition() {
    var options = {
        maximumAge: 3600000,
        timeout: 3000,
        enableHighAccuracy: true,
    }
    var watchID = navigator.geolocation.watchPosition(onSuccess, onError, options);

    function onSuccess(position) {
        alert('Latitude: ' + position.coords.latitude + '\n'
              'Longitude: ' + position.coords.longitude + '\n'
              'Altitude: ' + position.coords.altitude + '\n'
              'Accuracy: ' + position.coords.accuracy + '\n'
              'Altitude Accuracy: ' + position.coords.altitudeAccuracy + '\n'
              'Heading: ' + position.coords.heading + '\n'
              'Speed: ' + position.coords.speed + '\n'
              'Timestamp: ' + position.timestamp + '\n');
    }

    function onError(error) {
        alert('code: ' + error.code + '\n' + 'message: ' + error.message + '\n');
    }
}
In the example above, we are using two methods — `getCurrentPosition` and `watchPosition`. Both functions are using three parameters. Once we click the **CURRENT POSITION** button, the alert will show geolocation values.

If we click the **WATCH POSITION** button, the same alert will be triggered every three seconds. This way we can track movement changes of the user’s device.

**NOTE**

This plugin is using GPS. Sometimes it can’t return the values on time and the request will return `timeout` error. This is why we specified `enableHighAccuracy: true` and `maximumAge: 3600000`. This means that if a request isn’t completed on
time, we will use the last known value instead. In our example, we are setting maximumAge to 3600000 milliseconds.

Cordova - Globalization

This plugin is used for getting information about users’ locale language, date and time zone, currency, etc.

Step 1 - Installing Globalization Plugin

Open **command prompt** and install the plugin by typing the following code:

```
C:\Users\username\Desktop\CordovaProject>cordova plugin add cordova-plugin-globalization
```

Step 2 - Add Buttons

We will add several buttons to **index.html** to be able to call different methods that we will create later.

```
<button id="getLanguage">LANGUAGE</button>
<button id="getLocaleName">LOCALE NAME</button>
<button id="getDate">DATE</button>
<button id="getCurrency">CURRENCY</button>
```

Step 3 - Add Event Listeners

Event listeners will be added inside **getDeviceReady** function in **index.js** file to ensure that our app and Cordova are loaded before we start using it.

```
document.getElementById("getLanguage").addEventListener("click", getLanguage);
document.getElementById("getLocaleName").addEventListener("click", getLocaleName);
document.getElementById("getDate").addEventListener("click", getDate);
document.getElementById("getCurrency").addEventListener("click", getCurrency);
```

Step 4A - Language Function

The first function that we are using returns BCP 47 language tag of the client’s device. We will use **getPreferredLanguage** method. The function has two parameters **onSuccess** and **onError**. We are adding this function in **index.js**.

```
function getLanguage() {
    navigator.globalization.getPreferredLanguage(onSuccess, onError);
}
```
function onSuccess(language) {
    alert('language: ' + language.value + 'n');
}

function onError()
    alert('Error getting language');
}

Once we press the **LANGUAGE** button, the alert will be shown on screen.
Step 4B -Locale Function

This function returns BCP 47 tag for the client’s local settings. This function is similar as the one we created before. The only difference is that we are using `getLocaleName` method this time.

```javascript
function getLocaleName() {
    navigator.globalization.getLocaleName(onSuccess, onError);

    function onSuccess(locale) {
        alert('locale: ' + locale.value);
    }

    function onError() {
        alert('Error getting locale');
    }
}
```

When we click the **locale** button, the alert will show our locale tag.
Step 4C - Date Function

This function is used for returning date according to client’s locale and timezone setting. **date** parameter is the current date and **options** parameter is optional.

```javascript
function getDate() {
    var date = new Date();

    var options = {
        formatLength: 'short',
        selector: 'date and time'
    }
}
```
navigator.globalization.dateToString(date, onSuccess, onError, options);

function onSuccess(date) {
    alert('date: ' + date.value);
}

function onError() {
    alert('Error getting dateString');
}

We can now run the app and press **DATE** button to see the current date.
The last function that we will show is returning currency values according to client’s device settings and ISO 4217 currency code. You can see that the concept is the same.

```javascript
function getCurrency() {
    var currencyCode = 'EUR';
    navigator.globalization.getCurrencyPattern(currencyCode, onSuccess, onError);
}

function onSuccess(pattern) {
    alert('pattern: ' + pattern.pattern + '
    ' + pattern.code + '
    code: ' + pattern.code + '
    currency: ' + currencyCode + '
    date: 12/4/2015 5:59 PM');
}
```
The `CURRENCY` button will trigger alert that will show users currency pattern.

```javascript
function onError()
    alert('Error getting pattern');
}
```
This plugin offers other methods. You can see all of it in the table below.

<table>
<thead>
<tr>
<th>method</th>
<th>parameters</th>
<th>details</th>
</tr>
</thead>
<tbody>
<tr>
<td>getPreferredLanguage</td>
<td>onSuccess, onError</td>
<td>Returns client’s current language.</td>
</tr>
<tr>
<td>getLocaleName</td>
<td>onSuccess, onError</td>
<td>Returns client’s current locale settings.</td>
</tr>
<tr>
<td>dateToString</td>
<td>date, onSuccess, onError, options</td>
<td>Returns date according to client’s locale and timezone.</td>
</tr>
<tr>
<td>Function</td>
<td>Parameters</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>dateStringTo</td>
<td>dateString, onSuccess, onError, options</td>
<td>Parses a date according to client’s settings.</td>
</tr>
<tr>
<td>getCurrencyPattern</td>
<td>currencyCode, onSuccess, onError</td>
<td>Returns client’s currency pattern.</td>
</tr>
<tr>
<td>getDatePattern</td>
<td>onSuccess, onError, options</td>
<td>Returns client’s date pattern.</td>
</tr>
<tr>
<td>getDateNames</td>
<td>onSuccess, onError, options</td>
<td>Returns an array of names of the months, weeks or days according to client’s settings.</td>
</tr>
<tr>
<td>isDayLightSavingsTime</td>
<td>date, successCallback, errorCallback</td>
<td>Used to determine if the daylight saving time is active according to client’s time zone and calendar.</td>
</tr>
<tr>
<td>getFirstDayOfWeek</td>
<td>onSuccess, onError</td>
<td>Returns the first day of the week according to client settings.</td>
</tr>
<tr>
<td>numberToString</td>
<td>number, onSuccess, onError, options</td>
<td>Returns number according to client’s settings.</td>
</tr>
<tr>
<td>stringToNumber</td>
<td>string, onSuccess, onError, options</td>
<td>Parses a number according to client’s settings.</td>
</tr>
<tr>
<td>getNumberPattern</td>
<td>onSuccess, onError, options</td>
<td>Returns the number pattern according to client’s settings.</td>
</tr>
</tbody>
</table>

**Cordova - InAppBrowser**

This plugin is used for opening web browser inside Cordova app.

**Step 1 - Installing Plugin**

We need to install this plugin in **command prompt** window before we are able to use it.

```
C:\Users\username\Desktop\CordovaProject>cordova plugin add cordova-plugin-inappbrowser
```

**Step 2 - Add button**
We will add one button that will be used for opening `inAppBrowser` window in `index.html`.

**Step 3 - Add Event Listener**

Now let's add event listener for our button in `onDeviceReady` function in `index.js`.

```javascript
document.getElementById("openBrowser").addEventListener("click", openBrowser);
```

**Step 4 - Create Function**

In this step we are creating function that will open browser inside our app. We are assigning it to the `ref` variable that we can use later to add event listeners.

```javascript
function openBrowser() {
    var url = 'https://cordova.apache.org';
    var target = '_blank';
    var options = "location = yes"
    var ref = cordova.InAppBrowser.open(url, target, options);

    ref.addEventListener('loadstart', loadstartCallback);
    ref.addEventListener('loadstop', loadstopCallback);
    ref.addEventListener('loaderror', loaderrorCallback);
    ref.addEventListener('exit', exitCallback);

    function loadstartCallback(event) {
        console.log('Loading started: ' + event.url)
    }

    function loadstopCallback(event) {
        console.log('Loading finished: ' + event.url)
    }

    function loaderrorCallback(error) {
        console.log('Loading error: ' + error.message)
    }
}
```
function exitCallback() {
    console.log('Browser is closed...')
}

If we press **BROWSER** button, we will see the following output on screen.

Console will also listen to events. **loadstart** event will fire when URL is started loading and **loadstop** will fire when URL is loaded. We can see it in console.
Once we close the browser, **exit** event will fire.

There are other possible options for InAppBrowser window. We will explain it in the table below.

<table>
<thead>
<tr>
<th>S.No</th>
<th><strong>option &amp; details</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>location</strong> &lt;br&gt;Used to turn the browser location bar on or off. Values are <strong>yes</strong> or <strong>no</strong>.</td>
</tr>
<tr>
<td>2</td>
<td><strong>hidden</strong> &lt;br&gt;Used to hide or show inAppBrowser. Values are <strong>yes</strong> or <strong>no</strong>.</td>
</tr>
<tr>
<td>3</td>
<td><strong>clearCache</strong> &lt;br&gt;Used to clear browser cookie cache. Values are <strong>yes</strong> or <strong>no</strong>.</td>
</tr>
<tr>
<td>4</td>
<td><strong>clearsessioncache</strong> &lt;br&gt;Used to clear session cookie cache. Values are <strong>yes</strong> or <strong>no</strong>.</td>
</tr>
<tr>
<td>5</td>
<td><strong>zoom</strong> &lt;br&gt;Used to hide or show Android browser’s zoom controls. Values are <strong>yes</strong> or <strong>no</strong>.</td>
</tr>
<tr>
<td>6</td>
<td><strong>hardwareback</strong> &lt;br&gt;<strong>yes</strong> to use the hardware back button to navigate back through the browser history. <strong>no</strong> to close the browser once back button is clicked.</td>
</tr>
</tbody>
</table>

We can use **ref** (reference) variable for some other functionalities. We will show you just quick examples of it. For removing event listeners we can use –

```javascript
ref.removeEventListener(eventName, callback);
```

For closing InAppBrowser we can use –

```javascript
InAppBrowser.close();
```
If we opened hidden window, we can show it –

Even JavaScript code can be injected to the InAppBrowser –

Even JavaScript code can be injected to the InAppBrowser –

The same concept can be used for injecting CSS –

**Cordova - Media**

Cordova media plugin is used for recording and playing audio sounds in Cordova apps.

**Step 1 - Installing Media Plugin**

Media plugin can be installed by running the following code in `command prompt` window.

```
C:\Users\username\Desktop\CordovaProject>cordova plugin add cordova-plugin-media
```

**Step 2 - Add Buttons**

In this tutorial, we will create simple audio player. Let’s create buttons that we need in `index.html`.

```
<button id = "playAudio">PLAY</button>
<button id = "pauseAudio">PAUSE</button>
<button id = "stopAudio">STOP</button>
<button id = "volumeUp">VOLUME UP</button>
<button id = "volumeDown">VOLUME DOWN</button>
```

**Step 3 - Add Event Listeners**

Now we need to add event listeners for our buttons inside `onDeviceReady` function inside `index.js`.

```
document.getElementById("playAudio").addEventListener("click", playAudio);
document.getElementById("pauseAudio").addEventListener("click", pauseAudio);
document.getElementById("stopAudio").addEventListener("click", stopAudio);
```
Step 4A - Play Function

The first function that we are going to add is `playAudio`. We are defining `myMedia` outside of the function because we want to use it in functions that are going to be added later (pause, stop, volumeUp and volumeDown). This code is placed in `index.js` file.

```javascript
var myMedia = null;

function playAudio() {
    var src = "/android_asset/www/audio/piano.mp3";

    if (myMedia === null) {
        myMedia = new Media(src, onSuccess, onError);
    }

    function onSuccess() {
        console.log("playAudio Success");
    }

    function onError(error) {
        console.log("playAudio Error: " + error.code);
    }

    myMedia.play();
}
```

We can click **PLAY** button to start the piano music from the `src` path.

Step 4B - Pause and Stop Functions

The next functions that we need is `pauseAudio` and `stopAudio`.

```javascript
function pauseAudio() {
    if (myMedia) {
```
myMedia.pause();

function stopAudio() {
    if(myMedia) {
        myMedia.stop();
    }
    myMedia = null;
}

Now we can pause or stop the piano sound by clicking **PAUSE** or **STOP** buttons.

**Step 4C - Volume Functions**

To set the volume, we can use **setVolume** method. This method takes parameter with values from 0 to 1. We will set starting value to 0.5.

```javascript
var volumeValue = 0.5;

function volumeUp() {
    if(myMedia && volumeValue < 1) {
        myMedia.setVolume(volumeValue += 0.1);
    }
}

function volumeDown() {
    if(myMedia && volumeValue > 0) {
        myMedia.setVolume(volumeValue -= 0.1);
    }
}
```

Once we press **VOLUME UP** or **VOLUME DOWN** we can change the volume value by 0.1.

The following table shows other methods that this plugin provides.
<table>
<thead>
<tr>
<th>S.No</th>
<th>Method &amp; Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>getCurrentPosition</code></td>
</tr>
<tr>
<td></td>
<td>Returns current position of an audio.</td>
</tr>
<tr>
<td>2</td>
<td><code>getDuration</code></td>
</tr>
<tr>
<td></td>
<td>Returns duration of an audio.</td>
</tr>
<tr>
<td>3</td>
<td><code>play</code></td>
</tr>
<tr>
<td></td>
<td>Used for starting or resuming audio.</td>
</tr>
<tr>
<td>4</td>
<td><code>pause</code></td>
</tr>
<tr>
<td></td>
<td>Used for pausing audio.</td>
</tr>
<tr>
<td>5</td>
<td><code>release</code></td>
</tr>
<tr>
<td></td>
<td>Releases the underlying operating system’s audio resources.</td>
</tr>
<tr>
<td>6</td>
<td><code>seekTo</code></td>
</tr>
<tr>
<td></td>
<td>Used for changing position of an audio.</td>
</tr>
<tr>
<td>7</td>
<td><code>setVolume</code></td>
</tr>
<tr>
<td></td>
<td>Used for setting volume for audio.</td>
</tr>
<tr>
<td>8</td>
<td><code>startRecord</code></td>
</tr>
<tr>
<td></td>
<td>Start recording an audio file.</td>
</tr>
<tr>
<td>9</td>
<td><code>stopRecord</code></td>
</tr>
<tr>
<td></td>
<td>Stop recording an audio file.</td>
</tr>
<tr>
<td>10</td>
<td><code>stop</code></td>
</tr>
<tr>
<td></td>
<td>Stop playing an audio file.</td>
</tr>
</tbody>
</table>
Cordova - Media Capture

This plugin is used for accessing device's capture options.

Step 1 - Installing Media Capture Plugin

To install this plugin, we will open command prompt and run the following code:

```
C: \Users\username\Desktop\CordovaProject>cordova plugin add cordova-plugin-media-capture
```

Step 2 - Add Buttons

Since we want to show you how to capture audio, image and video, we will create three buttons in `index.html`.

```
<button id = "audioCapture">AUDI</button>
<button id = "imageCapture">IMAGE</button>
<button id = "videoCapture">VIDEO</button>
```

Step 3 - Add Event Listeners

The next step is adding event listeners inside `onDeviceReady` in `index.js`.

```
document.getElementById("audioCapture").addEventListener("click", audioCapture);
document.getElementById("imageCapture").addEventListener("click", imageCapture);
document.getElementById("videoCapture").addEventListener("click", videoCapture);
```

Step 4A - Capture Audio Function

The first callback function in `index.js` is `audioCapture`. To start sound recorder, we will use `captureAudio` method. We are using two options - `limit` will allow recording only one audio clip per single capture operation and `duration` is number of seconds of a sound clip.

```javascript
function audioCapture() {
    var options = {
        limit: 1,
        duration: 10
    };
    navigator.device.capture.captureAudio(onSuccess, onError, options);
}
```
When we press **AUDIO** button, sound recorder will open.

```javascript
function onSuccess(mediaFiles) {
   var i, path, len;
   for (i = 0, len = mediaFiles.length; i < len; i += 1) {
      path = mediaFiles[i].fullPath;
      console.log(mediaFiles);
   }
}

function onError(error) {
   navigator.notification.alert('Error code: ' + error.code, null, 'Capture Error');
}
```

When we press **AUDIO** button, sound recorder will open.
Console will show returned array of objects that users captured.

```
[MediaFile]
  
  0: MediaFile
    end: 0
    fullPath: "file:/storage/emulated/0/recording630789551.3gpp"
    lastModifiedDate: null
    lastModified: 1449789663000
    localURL: "file://localhost/sdcard/recording630789551.3gpp"
    name: "recording630789551.3gpp"
    size: 8416
    start: 0
    type: "audio/3gpp"
    __proto__: util.extend.F
    length: 1
    __proto__: Array[0]
```

**Step 4B - Capture Image Function**

THANKS FOR READING – VISIT OUR WEBSITE  www.educatererindia.com
The function for capturing image will be the same as the last one. The only difference is that we are using `captureImage` method this time.

```javascript
function imageCapture() {
    var options = {
        limit: 1
    };
    navigator.device.capture.captureImage(onSuccess, onError, options);

    function onSuccess(mediaFiles) {
        var i, path, len;
        for (i = 0, len = mediaFiles.length; i < len; i += 1) {
            path = mediaFiles[i].fullPath;
            console.log(mediaFiles);
        }
    }

    function onError(error) {
        navigator.notification.alert(`Error code: ' + error.code, null, 'Capture Error');
    }
}
```

Now we can click **IMAGE** button to start the camera.
When we take a picture, the console will log the array with an image object.

```javascript
[MediaFile]
  
  0: Mediafile
    
    end: 0
    
    fullPath: "file:/storage/emulated/0/DCIM/Camera/1449791189275.jpg"
    
    lastModifiedDate: null
    
    localURL: "file:///android_asset/localhost/sdcard/DCIM/Camera/1449791189275.jpg"
    
    name: "1449791189275.jpg"
    
    size: 7490
    
    start: 0
    
    type: "image/jpeg"
    
    __proto__: util.extend.F
    
    length: 1
    
    __proto__: Array[0]
```
Step 4C - Capture Video Function

Let’s repeat the same concept for capturing video. We will use `videoCapture` method this time.

```javascript
function videoCapture() {
    var options = {
        limit: 1,
        duration: 10
    };
    navigator.device.capture.captureVideo(onSuccess, onError, options);

    function onSuccess(mediaFiles) {
        var i, path, len;
        for (i = 0, len = mediaFiles.length; i < len; i += 1) {
            path = mediaFiles[i].fullPath;
            console.log(mediaFiles);
        }
    }

    function onError(error) {
        navigator.notification.alert('Error code: ' + error.code, null, 'Capture Error');
    }
}
```

If we press **VIDEO** button, the camera will open and we can record the video.
Once the video is saved, the console will return array once more. This time with video object inside.
Cordova - Network Information

This plugin provides information about device's network.

Step 1 - Installing Network Information Plugin

To install this plugin, we will open command prompt and run the following code:

```bash
C:\Users\username\Desktop\CordovaProject>cordova plugin add cordova-plugin-network-information
```

Step 2 - Add Buttons

Let's create one button in index.html that will be used for getting info about network.

```html
<button id = "networkInfo">INFO</button>
```

Step 3 - Add Event Listeners

We will add three event listeners inside onDeviceReady function in index.js. One will listen for clicks on the button we created before and the other two will listen for changes in connection status.

```javascript
document.getElementById("networkInfo").addEventListener("click", networkInfo);
document.addEventListener("offline", onOffline, false);
document.addEventListener("online", onOnline, false);
```

Step 4 - Creating Functions

`networkInfo` function will return info about current network connection once button is clicked. We are calling type method. The other functions are onOffline and onOnline. These functions are listening to the connection changes and any change will trigger the corresponding alert message.

```javascript
function networkInfo() {
    var networkState = navigator.connection.type;
    var states = {};
    states[Connection.UNKNOWN] = 'Unknown connection';
    states[Connection.ETHERNET] = 'Ethernet connection';
    states[Connection.WIFI] = 'WiFi connection';
    states[Connection.CELL_2G] = 'Cell 2G connection';
    ...
}
```
states[Connection.CELL_3G] = 'Cell 3G connection';
states[Connection.CELL_4G] = 'Cell 4G connection';
states[Connection.CELL] = 'Cell generic connection';
states[Connection.NONE] = 'No network connection';
alert('Connection type: ' + states[networkState]);

function onOffline() {
    alert('You are now offline!');
}

function onOnline() {
    alert('You are now online!');
}

When we start the app connected to the network, **onOnline** function will trigger alert.
If we press **INFO** button the alert will show our network state.
If we disconnect from the network, **onOffline** function will be called.
Cordova - Splash Screen

This plugin is used to display a splash screen on application launch.

Step 1 - Installing Splash Screen Plugin

Splash screen plugin can be installed in command prompt window by running the following code.

```
C:\Users\username\Desktop\CordovaProject>cordova plugin add cordova-plugin-splashscreen
```

Step 2 - Add Splash Screen

Adding splash screen is different from adding the other Cordova plugins. We need to open config.xml and add the following code snippets inside the widget element.
First snippet is **SplashScreen**. It has `value` property which is the name of the images in `platform/android/res/drawable-` folders. Cordova offers default `screen.png` images that we are using in this example, but you will probably want to add your own images. Important thing is to add images for portrait and landscape view and also to cover different screen sizes.

```xml
<preference name="SplashScreen" value="screen"/>
```

Second snippet we need to add is **SplashScreenDelay**. We are setting `value` to **3000** to hide the splash screen after three seconds.

```xml
<preference name="SplashScreenDelay" value="3000"/>
```

The last preference is optional. If value is set to **true**, the image will not be stretched to fit screen. If it is set to **false**, it will be stretched.

```xml
<preference name="SplashMaintainAspectRatio" value="true"/>
```

Now when we run the app, we will see the splash screen.
Cordova - Vibration

This plugin is used for connecting to device’s vibration functionality.

Step 1 - Installing Vibration Plugin

We can install this plugin in command prompt window by running the following code –

C:\Users\username\Desktop\CordovaProject>cordova plugin add cordova-plugin-vibration

Step 2 - Add Buttons

Once the plugin is installed we can add buttons in index.html that will be used later to trigger the vibration.
<button id = "vibration">VIBRATION</button>
<button id = "vibrationPattern">PATTERN</button>

Step 3 - Add Event Listeners
Now we are going to add event listeners inside `onDeviceReady` in `index.js`.

```javascript
document.getElementById("vibration").addEventListener("click", vibration);
document.getElementById("vibrationPattern").addEventListener("click", vibrationPattern);
```

Step 4 - Create Functions
This plugin is very easy to use. We will create two functions.

```javascript
function vibration() {
    var time = 3000;
    navigator.vibrate(time);
}

function vibrationPattern() {
    var pattern = [1000, 1000, 1000, 1000];
    navigator.vibrate(pattern);
}
```

The first function is taking time parameter. This parameter is used for setting the duration of the vibration. Device will vibrate for three seconds once we press VIBRATION button.

The second function is using pattern parameter. This array will ask device to vibrate for one second, then wait for one second, then repeat the process again.

Cordova - Whitelist
This plugin allows us to implement whitelist policy for app's navigation. When we create a new Cordova project, the whitelist plugin is installed and implemented by default. You can open the config.xml file to see allow-intent default settings provided by Cordova.

Navigation Whitelist
In the simple example below we are allowing links to some external URL. This code is placed in config.xml. Navigation to file:// URLs is allowed by default.

```xml
<allow-navigation href="http://example.com/*" />
```

The asterix sign, *, is used to allow navigation to multiple values. In the above example, we are allowing navigation to all sub-domains of the example.com. The same can be applied to protocol or prefix to the host.

```xml
<allow-navigation href="*:/*,example.com/*" />
```

**Intent Whitelist**

There is also the allow-intent element which is used to specify which URLs are allowed to open the system. You can see in the config.xml that Cordova already allowed most of the needed links for us.

**Network Request Whitelist**

When you look inside config.xml file, there is `access origin="*" />` element. This element allows all network requests to our app via Cordova hooks. If you want to allow only specific requests, you can delete it from the config.xml and set it yourself. The same principle is used as in previous examples.

```xml
<access origin = "http://example.com" />
```

This will allow all network requests from http://example.com.

**Content Security Policy**

You can see the current security policy for your app inside the head element in index.html.

```html
<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 ‘*’>
```

This is default configuration. If you want to allow everything from the same origin and example.com, then you can use –

```html
<meta http-equiv = "Content-Security-Policy" content = "default-src ‘self foo.com’”>
```

You can also allow everything, but restrict CSS and JavaScript to the same origin.
Since this is a beginners’ tutorial, we are recommending the default Cordova options. Once you get familiar with Cordova, you can try some different values.

**Cordova - Best Practices**

Cordova is used for creating hybrid mobile apps, so you need to consider this before you choose it for your project. Below are the best practices for Cordova apps development.

**Single Page Apps**

This is the recommended design for all Cordova apps. SPA is using client-side router and navigation loaded on the single page (usually `index.html`). The routing is handled via AJAX. If you have followed our tutorials, you probably noticed that almost every Cordova plugin needs to wait until the device is ready before it can be used. SPA design will improve loading speed and overall performance.

**Touch Events**

Since Cordova is used for mobile world it is natural to use `touchstart` and `touchend` events instead of `click` events. The click events have 300ms delay, so the clicks don’t feel native. On the other hand, touch events aren’t supported on every platform. You should take this into consideration before you decide what to use.

**Animations**

You should always use hardware accelerated **CSS Transitions** instead of JavaScript animations since they will perform better on mobile devices.

**Storage**

Use storage caching as much as possible. Mobile network connections are usually bad, so you should minimize network calls inside your app. You should also handle offline status of the app, since there will be times when user’s devices are offline.

**Scrolling**

Most of the time the first slow part inside your app will be scrolling lists. There are couple of ways to improve scrolling performance of the app. Our recommendation
is to use native scrolling. When there are lots of items in the list, you should load them partially. Use loaders when necessary.

Images
Images can also slow the mobile app. You should use CSS image sprites whenever possible. Try to fit the images perfectly instead of scaling it.

CSS styles
You should avoid shadows and gradients, since they slow the rendering time of the page.

Simplification
Browser’s DOM is slow, so you should try to minimize DOM manipulation and number of DOM elements.

Testing
Ensure that you test your app on as many devices and operating system versions as possible. If app works flawlessly on one device, it doesn’t necessary mean that it will work on some other device or platform.