3

Working with Audio, Images, and Video

In this chapter, we will cover:

- Capturing audio using the device audio recording application
- Recording audio within your application
- Playing audio files from the local filesystem or over HTTP
- Capturing video using the device video recording application
- Loading a photograph from the device camera roll/library
- Applying an effect to an image using canvas

Introduction

This chapter will include a number of recipes that outline the functionality required to capture audio, video, and camera data, as well as the playback of audio files from the local system and remote host. We will also have a look at how to use the HTML5 canvas element to edit an image on the fly.

Capturing audio using the devices audio recording application

The PhoneGap API allows developers the ability to interact with the audio recording application on the device and save the recorded audio file for later use.

How to do it...

We'll make use of the `Capture` object and the `captureAudio` method it contains to invoke the native device audio recording application to record our audio. The following steps will help you to do so:

1. Create the initial layout for the application, and include the JavaScript references to jQuery and Cordova. We'll also set a stylesheet reference pointing to `style.css`.

```
<!DOCTYPE html>
<html>
  <head>
    <meta name="viewport" content="user-scalable=no,
         initial-scale=1, maximum-scale=1,
         minimum-scale=1, width=device-width;" />
    <title>Audio Capture</title>
    <link rel="stylesheet" href="style.css" />
    <script src="jquery/jquery-1.8.0.min.js"></script>
    <script type="text/javascript"
         src="cordova-2.0.0.js"></script>
</head>
<body>



</body>
</html>
```

2. Create a new `button` element within the `body` tags of the document, and set the `id` attribute to `record`. We'll use this to bind a touch handler to it:

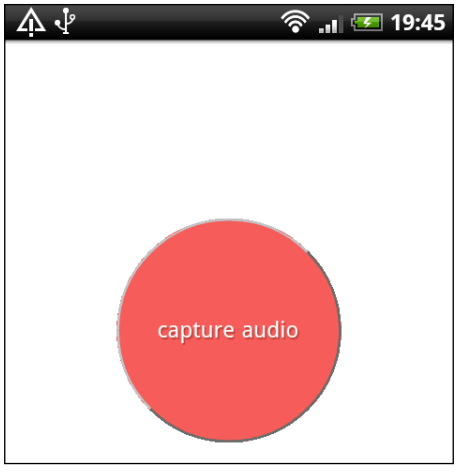
```
<button id="record">capture audio</button>
```

3. Create a new file called `style.css` and include some CSS to format the `button` element:

```
#record {
  display: block;
  padding: .4em .8em;
  text-decoration: none;
  text-shadow: 1px 1px 1px rgba(0,0,0,.3);
  -webkit-transition:.3s -webkit-box-shadow, .3s padding;
  transition:.3s box-shadow, .3s padding;
  border-radius: 200px;
  background: rgba(255,0,0,.6);
  width: 10em;
  height: 10em;
  color: white;
```

```
  position: absolute;
  top: 25%;
  left: 25%;
}
```

4. With the user interface added to the page and the styles applied, the application looks something like the following screenshot:



5. Now let's start adding our custom code. Create a new `script` tag block before the closing `head` tag. Within this we'll set up an event listener, which will call the `onDeviceReady` method once the PhoneGap code is ready to run.
6. We'll also create a global variable called `audioCapture`, which will hold our `capture` object:

```
<script type="text/javascript">

    document.addEventListener("deviceready",
     onDeviceReady, true);

    var audioCapture = '';

</script>
```

7. We now need to create the `onDeviceReady` method. This will firstly assign the `capture` object to the variable we defined earlier. We'll also bind a `touchstart` event to the `button` element, which when pressed will run the `getAudio` method to commence the capture process:

```
function onDeviceReady() {
   audioCapture = navigator.device.capture;
```

```
    $('#record').bind('touchstart', function() {
        getAudio();
});
}
```

8. To begin the audio capture, we need to call the `captureAudio()` method from the global `capture` object. This function accepts three parameters. The first is the name of the method to run after a successful transaction. The second is the name of the error handler method to run if we encounter any problems trying to obtain audio. The third is an array of configuration options for the capture request.

9. In this example we are forcing the application to retrieve only one audio capture, which is also the default value:

```
function getAudio() {
audioCapture.captureAudio(
    onSuccess,
    onError,
    {limit: 1}
);
}
```

10. Following on from a successful transaction, we will receive an array of objects containing details for each audio file that was captured. We'll loop over that array and generate a string containing all of the properties for each file, which we'll insert in to the DOM before the `button` element.

```
function onSuccess(audioObject) {
var i, output = '';
for (i = 0; i < audioObject.length; i++) {
    output += 'Name: ' + audioObject[i].name + '<br />' +
          'Full Path: ' + audioObject[i].fullPath + '<br />' +
          'Type: ' + audioObject[i].type + '<br />' +
          'Created: '
+ new Date(audioObject[i].lastModifiedDate) + '<br />' +
          'Size: ' + audioObject[i].size + '<br />========';
   }
$('#record').before(output);
}
```

11. If we encountered an error during the process, the `onError` method will fire. The method will provide us with access to an error object, which contains the code for the error. We can use a switch statement here to customize the message that we will return to our user.

```
function onError(error) {
var errReason;
   switch(error.code) {
```

```
        case 0:
          errReason = 'The microphone failed to capture sound.';
    break;
        case 1:
            errReason = 'The audio capture application is currently
busy with another request.';
        break;
        case 2:
            errReason = 'An invalid parameter was sent to the
API.';
        break;
        case 3:
            errReason = 'You left the audio capture application
without recording anything.';
        break;
        case 4:
            errReason = 'Your device does not support the audio
capture request.';
        break;
    }
    alert('The following error occurred: ' + errReason);
}
```

12. If we run our application and press the button, the device's default audio recording application will open and we can record our audio.

13. Once we have finished recording, our application will receive the audio data from the callback method and the output will look like the following screenshot:

How it works...

The `Capture` object available through the PhoneGap API allows us to access the media capture capabilities of the device. By specifying the media type we wish to capture by calling the `captureAudio` method, an asynchronous call is made to the device's native audio recording application.

In this example we requested the capture of only one audio file. Setting the limit value within the optional configuration to a value greater than one can alter this.

The request is finished when one of two things happen:

- The maximum number of recordings has been created
- The user exits the native audio recording application

Following a successful callback from the request operation, we receive an array of objects that contains properties for each individual media file, which contains the following properties we can read:

- `name`: A `DOMString` object that contains the name of the file
- `fullPath`: A `DOMString` object that contains the full path of the file
- `type`: A `DOMString` object that includes the mime type of the returned media file
- `lastModifiedTime`: A `Date` object that contains the date and time that the file was last modified
- `size`: A `Number` value that contains the size of the file in bytes

To find out more about the `captureAudio` capabilities offered by the PhoneGap API, check out the official documentation here: `http://docs.phonegap.com/en/2.0.0/cordova_media_capture_capture.md.html#capture.captureAudio`.

See also

- The *Playing audio files from the local filesystem or over HTTP* recipe

Recording audio within your application

The PhoneGap API provides us with the ability to record audio directly within our application, bypassing the native audio recording application.

How to do it...

We will use the `Media` object to create a reference to an audio file into which we'll record the audio data.

1. Create the initial layout for your HTML page. This will include the references to the jQuery and jQuery UI JavaScript libraries, the style sheets, and the Cordova JavaScript library. We'll also include an empty `script` tag block which will contain our custom code. This is shown in the following code block:

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta name="viewport" content="user-scalable=no,
         initial-scale=1, maximum-scale=1,
         minimum-scale=1, width=device-width;" />
    <title>Audio Recorder</title>
        <link rel="stylesheet" href="style.css" />
     <link type="text/css"
     href="jquery/css/smoothness/jquery-ui-1.8.23.custom.css"
     rel="stylesheet" />
        <script type="text/javascript"
        src="jquery/jquery-1.8.0.min.js"></script>
        <script type="text/javascript"
        src="jquery/jquery-ui-1.8.23.custom.min.js"></script>
        <script type="text/javascript"
                src="cordova-2.0.0.js"></script>
        <script type="text/javascript">
```

```
        </script>
</head>
<body>

</body>
</html>
```

2. Include three elements within the `body` of our application. The first is a `div` with the `id` attribute set to `progressbar`. The second is a `div` with the `id` attribute set to `message`, and the third is a `button` element with the `id` attribute set to `record`.

```
<div id="progressbar"></div>
<div id="message"></div>
<button id="record"></button>
```

3. Now let's start adding our custom code within the empty `script` tag block. We'll begin by defining some global variables that we'll use in the application. We'll also create the event listener to ensure the device is ready before we proceed.

4. The `onDeviceReady` function will then run a new function called `recordPrepare`, as shown in the following code:

```
var maxTime = 10,
    countdownInt = 3,
    src,
    audioRecording,
    stopRecording;

document.addEventListener("deviceready",
 onDeviceReady, false);

function onDeviceReady() {
    recordPrepare();
}
```

5. The `recordPrepare` button will be used more than once in our application to reset the state of the button to record audio. Here, we unbind any actions applied to the button, set the HTML value and bind the `touchstart` handler to run a function called `recordAudio`:

```
function recordPrepare() {
    $('#record').unbind();
    $('#record').html('Start recording');
    $('#record').bind('touchstart', function() {
        recordAudio();
    });
}
```

6. Let's now create the `recordAudio()` function, which will create the audio file. We'll switch the value and bind events applied to our button to allow the user to manually end the recording. We also set the `Media` object to a variable, `audioRecording`, and pass in the destination for the file in the form of the `src` parameter, as well as the success and error callback methods.

7. A `setInterval` method is included, which will count down from three to zero to give the user some time to prepare for the recording. When the countdown is complete, we then invoke the `startRecord` method from the `Media` object and start another `setInterval` method. This will count to ten and will automatically stop the recording when the limit has been reached.

```
function recordAudio() {

    $('#record').unbind();
    $('#record').html('Stop recording');
    $('#record').bind('touchstart', function() {
        stopRecording();
    });

    src = 'recording_' + Math.round(new Date().getTime()/1000) +
'.mp3';

    audioRecording = new Media(src, onSuccess, onError);

    var startCountdown = setInterval(function() {

        $('#message').html('Recording will start in ' +
countdownInt + ' seconds...');
        countdownInt = countdownInt -1;

        if(countdownInt <= 0) {
            countdownInt = 3;
            clearInterval(startCountdown);
            audioRecording.startRecord();

            var recTime = 0;
                recInterval = setInterval(function() {
                recTime = recTime + 1;

                $('#message').html(Math.round(maxTime - recTime) +
' seconds remaining...');
```

```
                var progPerc = 100-((100/maxTime) * recTime);
                setProgress(progPerc);
                if (recTime >= maxTime) {
                    stopRecording();
                }
            }, 1000);
        }
    }, 1000);
}
```

8. As our recording is underway we can update the progress bar using the jQuery UI library and set it to the current value to show how much time is remaining.

```
function setProgress(progress) {
    $("#progressbar").progressbar({
            value: progress
    });
}
```

9. When a recording is stopped, we want to clear the interval timer and run the `stopRecord` method from the `Media` object. We'll also clear the value of the progress bar to zero, and reset the button bindings to prepare for the next recording.

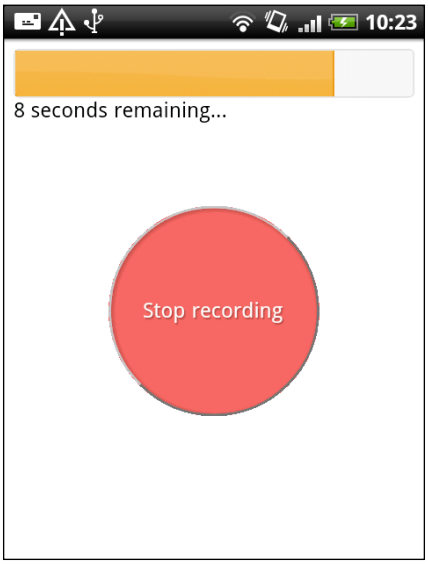
```
function stopRecording() {
    clearInterval(recInterval);
    audioRecording.stopRecord();
    setProgress(0);
    recordPrepare();
}
```

Finally we can add in our success and error callback methods:

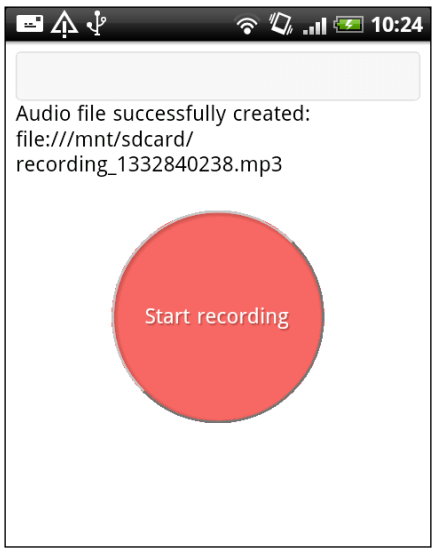
```
function onSuccess() {
    $('#message').html('Audio file successfully
    created:<br />' + src);
}

function onError(error) {
    $('#message').html('code: ' + error.code    + '\n' +
          'message: ' + error.message + '\n');
}
```

10. When we run the application to start recording, the output should look something like the following screenshot:



11. After a successful recording the user would be presented with the URI to the recorded file that we could use to access, upload, or play back the file, as shown in the following screenshot:



How it works...

The `Media` object has the ability to record and play back audio files. When choosing to utilize the object for recording, we need to provide the method with the URI for the destination file on the local device.

To start a recording we simply call the `startRecord` method from the `Media` object, and to stop the recording we need to call the `stopRecord` method.

To find out more about the available methods within the `Media` object, please refer to the official documentation, available here: `http://docs.phonegap.com/en/2.0.0/cordova_media_media.md.html#Media`.

See also

- The *Saving a file to device storage* recipe in *Chapter 2*, *File System, Storage, and Local Databases*
- The *Opening a local file from device storage* recipe in *Chapter 2*, *File System, Storage, and Local Databases*

Playing audio files from the local filesystem or over HTTP

The PhoneGap API provides us with a relatively straightforward process to play back audio files. These can be files stored within the application's local filesystem, bundled with the application, or over remote files accessible by a network connection. Wherever the files may be, the method of playback is achieved in exactly the same way.

How to do it...

We must create a new `Media` object and pass into it the location of the audio file we want to play back:

1. Create the initial layout for the HTML, and include the relevant references to the JavaScript and style sheets. In this example we are going to be using the jQuery Mobile framework (`http://jquerymobile.com/`):

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta name="viewport" content="width=screen.width; user-
```

```
scalable=no" />
    <title>Audio Player</title>
    <link rel="stylesheet" href="jquery/jquery.mobile-1.1.1.min.
css" type="text/css">
    <script type="text/javascript" src="jquery/jquery-1.8.0.min.
js"></script>
    <script type="text/javascript" src="jquery/jquery.mobile-
1.1.1.min.js"></script>
    <script type="text/javascript" src="cordova-2.0.0.js"></
script>

</head>
<body>

</body>
</html>
```

2. Create the layout for the application within the `body` tags. Here we are specifying a page for the jQuery Mobile framework, and four key `div` elements that have been assigned the role of buttons. We will reference their `id` attributes in our code, as follows:

```
<div data-role="page" id="page-home">
    <div data-role="header">
        <h1>PhoneGap Audio Player</h1>
    </div>

    <div data-role="content">

        <div data-role="button"
        id="playLocalAudio">Play Local Audio</div>
        <div data-role="button"
        id="playRemoteAudio">Play Remote Audio</div>
        <div data-role="button"
        id="pauseaudio">Pause Audio</div>
        <div data-role="button"
        id="stopaudio">Stop Audio</div>

        <div class="ui-grid-a">
            <div class="ui-block-a"> Current:
         <span id="audioPosition">0 sec</span></div>
            <div class="ui-block-b">Total:
         <span id="mediaDuration">0</span> sec</div>
        </div>
    </div>
</div>
```

3. Create a new `script` tag block within the `head` tag to contain our custom code, into which we'll add out an event listener to check that the device is ready to proceed, and also, some required global variables.

```
<script type="text/javascript">

document.addEventListener("deviceready", onDeviceReady, true);

var audioMedia = null,
  audioTimer = null,
  duration = -1,
  is_paused = false;


</script>
```

4. The `onDeviceReady` method binds a `touchstart` event to all four of our buttons in the main page content. For the local audio option, this example is set to read a file from the Android asset location. In both play functions, we pass the audio source to the `playAudio` method:

```
function onDeviceReady() {

  $("#playLocalAudio").bind('touchstart', function() {

    stopAudio();
    var srcLocal = '/android_asset/www/CFHour_Intro.mp3';
    playAudio(srcLocal);

  });

  $("#playRemoteAudio").bind('touchstart', function() {

    stopAudio();
    var srcRemote = 'http://traffic.libsyn.com/cfhour/Show_138_-_
ESAPI_StackOverflow_and_Community.mp3';
    playAudio(srcRemote);

  });

  $("#pauseaudio").bind('touchstart', function() {
      pauseAudio();
  });
```

```
    $("#stopaudio").bind('touchstart', function() {
        stopAudio();
    });

}
```

5. Now let's add in the `playAudio` method. This will firstly check to see if the `audioMedia` object has been assigned and we have an active audio file. If not, we will reset the duration and position values and create a new `Media` object reference, passing in the source of the audio file.
6. To update the duration and current position of the audio file, we will set a new interval timer which will check once every second and obtain these details from the `getCurrentPosition` and `getDuration` methods, available from the `Media` object:

```
function playAudio(src) {

    if (audioMedia === null) {
        $("#mediaDuration").html("0");
        $("#audioPosition").html("Loading...");
        audioMedia = new Media(src, onSuccess, onError);
        audioMedia.play();
    } else {
        if (is_paused) {
            is_paused = false;
            audioMedia.play();
        }
    }

    if (audioTimer === null) {
        audioTimer = setInterval(function() {
audioMedia.getCurrentPosition(
function(position) {
  if (position > -1) {

  setAudioPosition(Math.round(position));
    if (duration <= 0) {
   duration = audioMedia.getDuration();
        if (duration > 0) {
          duration = Math.round(duration);
       $("#mediaDuration").html(duration);
        }
    }
  }
},
```

```
function(error) {
  console.log("Error getting position=" + error);
  setAudioPosition("Error: " + error);
}
        );
        }, 1000);
    }
}
```

7. The `setAudioPosition` method will update the content in the `audioPosition` element with the current details.

```
function setAudioPosition(position) {
    $("#audioPosition").html(position + " sec");
}
```

8. Now we can include the two remaining methods assigned to the touch handlers to control pausing and stopping the audio playback.

```
function pauseAudio() {
    if (is_paused) return;
    if (audioMedia) {
        is_paused = true;
        audioMedia.pause();
    }
}

function stopAudio() {
    if (audioMedia) {
        audioMedia.stop();
        audioMedia.release();
        audioMedia = null;
    }
    if (audioTimer) {
        clearInterval(audioTimer);
        audioTimer = null;
    }

    is_paused = false;
    duration = 0;
}
```

9. Lastly, let's write the success and error callback methods. In essence, they both reset the values to default positions in preparation for the next playback request.

```
function onSuccess() {
    setAudioPosition(duration);
```

```
        clearInterval(audioTimer);
        audioTimer = null;
        audioMedia = null;
        is_paused = false;
        duration = -1;
    }

    function onError(error) {
        alert('code: ' + error.code + '\n' +
                'message: ' + error.message + '\n');
        clearInterval(audioTimer);
        audioTimer = null;
        audioMedia = null;
        is_paused = false;
        setAudioPosition("0");
    }
```

10. When we run the application on the device, the output will be similar to that shown in the following screenshot:

How it works...

The `Media` object has the ability to record and play back audio files. For media playback, we simply pass in the location of the audio file, remote or local, in to the Media instantiation call, along with the success and error handlers.

Playback is controlled by the `Media` objects' built-in methods available through the PhoneGap API.

To find out more about all of the available methods within the Media object, please refer to the official documentation, available here: `http://docs.phonegap.com/en/2.0.0/cordova_media_media.md.html#Media`.

There's more...

In this example, we are assuming the developer is building for an Android device, and so we have referenced the location of the local file using the `android_asset` reference. To cater for other device operating systems you can use the `Device` object available in the PhoneGap API to determine which platform is running the application. Using the response from this check, you can then create a switch statement to provide the correct path to the local file.

To find out more about the Device object, please refer to the official documentation, available here: `http://docs.phonegap.com/en/2.0.0/cordova_device_device.md.html#Device`.

Capturing video using the devices video recording application

The PhoneGap API provides us with the ability to easily access the native video recording application on the mobile device and save the captured footage.

How to do it...

We will use the `Capture` object and the `captureVideo` method it contains to invoke the native video recording application.

1. Create the initial layout for the application, and include the JavaScript references to jQuery and PhoneGap. We'll also set a stylesheet reference pointing to `style.css`:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Video Capture</title>
    <link rel="stylesheet" href="style.css" />
    <script type="text/javascript"
            src="jquery/jquery-1.8.0.min.js"></script>
    <script type="text/javascript"
         src="cordova-2.0.0.js"></script>
```

```
</head>
<body>


</body>
</html>
```

2. Create a new `button` element within the `body` tags of the document, and set the `id` attribute to `record`. We'll use this to bind a touch handler to it.

```
<button id="record">capture video</button>
```

3. Now let's start to add our custom code. Create a new `script` tag block before the closing `head` tag. Within this we'll set up our event listener, which will call the `onDeviceReady` method once the PhoneGap code is ready to run.
4. We'll also create a global variable called `videoCapture`, which will hold our `capture` object.

```
<script type="text/javascript">

    document.addEventListener("deviceready",
     onDeviceReady, true);

    var videoCapture = '';

</script>
```

5. We now need to create the `onDeviceReady` method. This will firstly assign the `capture` object to the variable we defined earlier. We'll also bind a `touchstart` event to the `button` element, which when pressed will run the `getVideo` method to commence the capture process:

```
function onDeviceReady() {
   videoCapture = navigator.device.capture;

   $('#record').bind('touchstart', function() {
     getVideo();
});
}
```

6. To begin the video capture, we need to call the `captureVideo` method from the global `capture` object. This function accepts three parameters. The first is the name of the method to run after a successful transaction. The second is the name of the error handler method to run if we encounter any problems trying to obtain the video. The third is an array of configuration options for the capture request.

7. In this example we are requesting the application to retrieve two separate video captures, as shown in the following block of code:

```
function getVideo() {
videoCapture.captureVideo(
    onSuccess,
    onError,
    {limit: 2}
);
}
```

8. Following on from a successful transaction, we will receive an array of objects containing details for each video file that was captured. We'll loop over that array and generate a string containing all of the properties for each file, which we'll insert in to the DOM before the `button` element.

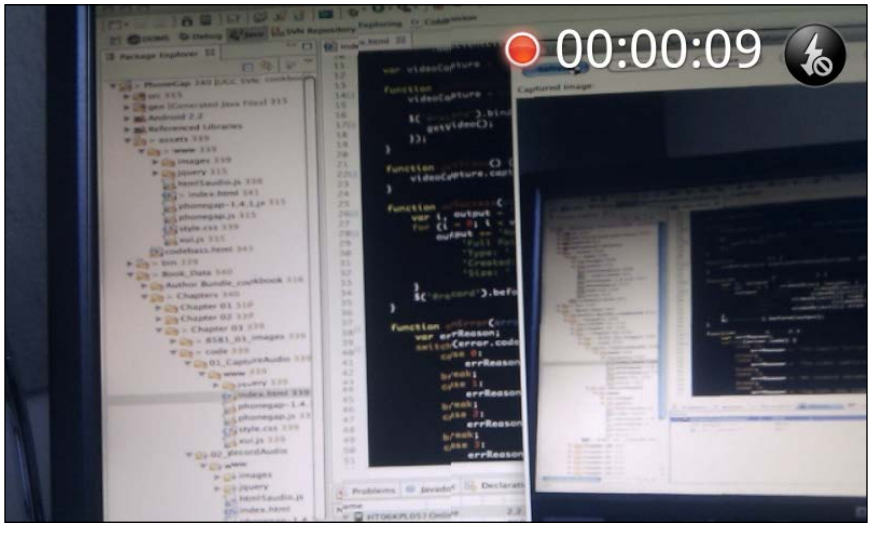
```
function onSuccess(videoObject) {
var i, output = '';
for (i = 0; i < videoObject.length; i += 1) {
    output += 'Name: ' + videoObject[i].name + '<br />' +
       'Full Path: ' + videoObject[i].fullPath + '<br />' +
         'Type: ' + videoObject[i].type + '<br />' +
         'Created: '
+ new Date(videoObject[i].lastModifiedDate) + '<br />' +
         'Size: ' + videoObject[i].size + '<br />========';
   }
$('#record').before(output);
}
```

9. If we encountered an error during the process, the `onError` method will fire. The method will provide us with access to an error object, which contains the code for the error. We can use a switch statement here to customize the message that we will return to our user, as follows:

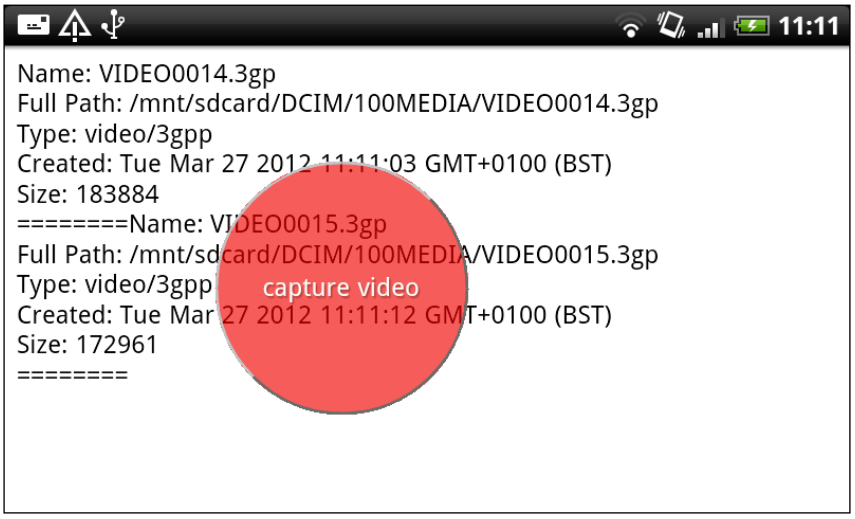
```
function onError(error) {
var errReason;
   switch(error.code) {
       case 0:
        errReason = 'The camera failed to capture video.';
    break;
       case 1:
           errReason = 'The video capture application is currently
busy with another request.';
```

```
        break;
        case 2:
            errReason = 'An invalid parameter was sent to the
API.';
        break;
        case 3:
            errReason = 'You left the video capture application
without recording anything.';
        break;
        case 4:
            errReason = 'Your device does not support the video
capture request.';
        break;
    }
    alert('The following error occurred: ' + errReason);
}
```

10. If we run our application and press the button, the device's default video recording application will open and we can record our video, as shown in the following screenshot:



11. Once we have finished recording, our application will receive the video data from the callback method and the output will look like the following screenshot:



How it works...

The `Capture` object available through the PhoneGap API allows us access to the media capture capabilities of the device. By specifying the media type we wish to capture by calling the `captureVideo` method, an asynchronous call is made to the device's native video recording application.

In this example, we forced the method to request two video captures by setting the limit property in the optional configuration options – the default value for the limit is set to one.

The request is finished when one of the two things happen:

- The maximum number of recordings has been created
- The user exits the native video recording application

Following a successful callback from the request operation, we receive an array of objects that contains properties for each individual media file, which contains the following properties, as we can read:

- `name`: A `DOMString` object that contains the name of the file
- `fullPath`: A `DOMString` object that contains the full path of the file
- `type`: A `DOMString` object that includes the mime type of the returned media file

- `lastModifiedTime`: A `Date` object that contains the date and time that the file was last modified
- `size`: A `Number` value that contains the size of the file in bytes

To find out more about the `captureVideo` capabilities offered by the PhoneGap API, check out the official documentation here: `http://docs.phonegap.com/en/2.0.0/cordova_media_capture_capture.md.html#capture.captureVideo`.

Loading a photograph from the devices camera roll/library

Different devices will store saved photographs in different locations, typically in either a photo library or saved photo album. The PhoneGap API gives developers the ability to select or specify from which location an image should be retrieved.

How to do it...

We must use the `getPicture` method available from the `Camera` object to select an image from either the device library or to capture a new image directly from the camera.

1. Create the initial layout for the HTML page, and include the required references to the jQuery and Cordova JavaScript libraries.

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta name="viewport" content="width=screen.width; user-
scalable=no" />
    <title>Photo Finder</title>
    <script type="text/javascript"
            src="jquery/jquery-1.8.0.min.js"></script>
    <script type="text/javascript"
            src="cordova-2.0.0.js"></script>
</head>
<body>

</body>
</html>
```

2. The `body` tag of our application will contain four elements. We'll need to provide two buttons, both with the `class` attribute set to `photo`, and each one with the `id` attribute set to `cameraPhoto` and `libraryPhoto` respectively.
3. Well also need to create a `div` element with `id` set to `message` and an `img` tag with `id` set to `image`, as shown in the following code block:

```
<button class="photo"
id="cameraPhoto">Take New Photo</button><br />
<button class="photo"
id="libraryPhoto">Select From Library</button><br />
<div id="message"></div><br />
<img id="image" />
```

4. Create a new `script` tag block within the head of the document and include the event listener that will fire when the PhoneGap native code is compiled and ready. Below this, create the `onDeviceReady` function, within which we'll apply a bind handler to the buttons by using the jQuery class selector.
5. Depending on the value of the selected button's `id` attribute, the `switch` statement will run the particular method to obtain the image.

```
<script type="text/javascript">

document.addEventListener("deviceready",onDeviceReady,false);

function onDeviceReady() {
$('.photo').bind('touchstart', function() {
  switch($(this).attr('id')) {
    case 'cameraPhoto':
      capturePhoto();
    break;
    case 'libraryPhoto':
      getPhoto();
    break;
  }
});
}


</script>
```

6. Let's now add the first of our image capture functions, `capturePhoto`. This calls the `getPicture` method from the `Camera` object. Here we are asking for the highest quality image returned and with a scaled image to match the provided sizes.

```
function capturePhoto() {
  navigator.camera.getPicture(onSuccess, onFail,
    {
   quality: 100,
      targetWidth: 250,
      targetHeight: 250
    }
  );
}
```

7. The second image capture method is `getPhoto`. In this method we once again call the `getPicture` method, but we now pass in the `sourceType` option value to request the image to be selected from the device photo library.

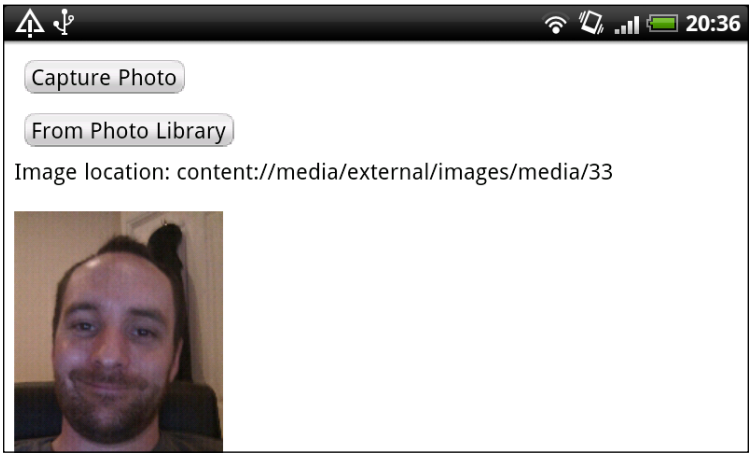
```
function getPhoto() {
  navigator.camera.getPicture(onSuccess, onFail,
    {
      quality: 100,
      destinationType: Camera.DestinationType.FILE_URI,
      sourceType: Camera.PictureSourceType.PHOTOLIBRARY,
      targetWidth: 250,
      targetHeight: 250
    }
  );
}
```

8. Finally, let's add in the success and error handlers, which both of our capture methods will use. The `onSuccess` method will display the returned image, setting it as the source for the `image` element:

```
function onSuccess(imageURI) {
  $('#image').attr('src', imageURI);
  $('#message').html('Image location: ' + imageURI);
}

function onFail(message) {
  $('#message').html(message);
}
```

9. When we run the application on the device, the output would look something like the following screenshot:



How it works...

The `Camera` object available through the PhoneGap API allows us to interact with the default camera application on the device. The `Camera` object itself contains just the one method, `getPicture`. Depending on the `sourceType` value being sent through in the capture request method, we can obtain the image from either the device camera or by selecting a saved image from the photo library or photo album.

In this example we retrieved the URI for the image to use as the source for an `img` tag. The method can also return the image as a Base64 encoded image, if requested.

There are a number of optional parameters that we can send in to the method calls to customize the camera settings. For detailed information about each parameter, please refer to the official documentation available here: `http://docs.phonegap.com/en/2.0.0/cordova_camera_camera.md.html#cameraOptions`.

There's more...

In this recipe, we requested that the images be scaled to match a certain dimension, while maintaining the aspect ratio. When selecting an image from the library or saved album, PhoneGap resizes the image and stores it in a temporary cache directory on the device. While this means resizing, it is as painless as we would want it to be, the image may not persist or will be overwritten when the next image is resized.

If you want to save the resized images in a permanent location after creating them, make sure you check out the recipes within this book on how to interact with the local file system and how to save files.

Now that we can easily obtain an image from the device, there are a number of things we can do with it. For an example, take a look at the next recipe in this book.

See also

- The *Uploading a file to a remote server via a POST request* recipe in *Chapter 2*, *File System, Storage, and Local Databases*

Applying an effect to an image using canvas

Capturing a photo on your device is fantastic, but what can we do with an image once we have it in our application? In this recipe, we'll create some simple functions to edit the color of an image without altering the original source.

How to do it...

We must create the use of the HTML5 canvas element to load and edit the values of a stored image, by performing the following steps:

1. Create the initial HTML layout and include the references to the jQuery and Cordova JavaScript files in the `head` tag of the document:

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta name="viewport" content="width=screen.width; user-
scalable=no" />
    <title>Image Effects</title>
    <script type="text/javascript"
            src="jquery/jquery-1.8.0.min.js"></script>
    <script type="text/javascript"
            src="cordova-2.0.0.js"></script>
</head>
<body>

</body>
</html>
```

2. Include a reference to the `rgb.js` file available in the downloadable code files of this book, below the Cordova JavaScript reference. This contains a required array of variables for one of our image manipulation functions.

```
<script type="text/javascript" src="cordova-2.0.0.js"></script>

<script type="text/javascript"
src="rgb.js"></script>


</head>
```

3. The `body` tag of our application will hold three `button` elements, each with a specific `id` attribute that we will reference within the custom code. We'll also need an `img` tag with the `id` attribute set to `sourceImage`, which will display the original image we want to manipulate.
4. Finally, we need to include a `canvas` element with the `id` attribute set to `myCanvas`, as shown in the following code block:

```
<button id="grayscale">Grayscale</button>
<button id="sepia">Sepia</button>
<button id="reset">Reset</button><br />

<img id="sourceImage" src="awesome.jpg"
alt="source image" height="150" width="150" />
<canvas id="myCanvas" width="300" height="300"></canvas>
```

5. Let's start to add our custom code. Create a new `script` tag block before the closing `head` tag, into which we'll add our event listener to ensure that PhoneGap is fully loaded before we proceed. We'll also create some required global variables.

```
<script type="text/javascript">

  document.addEventListener(
    "deviceready", onDeviceReady, true);

var canvas,
      context,
      image,
      imgObj,
      noise = 20;

</script>
```

6. Create the `onDeviceReady` method, which will run once the native code is ready. Here, we firstly want to run a method called `reset` which will restore our `canvas` to its default source. We'll also bind the `touchstart` handlers to our three buttons, each of which will run their own methods.

```
function onDeviceReady() {
  reset();

  $('#grayscale').bind('touchstart', function() {
    grayscaleImage();
  });

  $('#sepia').bind('touchstart', function() {
    processSepia();
  });

  $('#reset').bind('touchstart', function() {
    reset();
  });
}
```

7. The `reset` method creates the `canvas` reference and its context, and applies the source from our starting image into it.

```
function reset() {
  canvas = document.getElementById('myCanvas');
  context = canvas.getContext("2d");
  image = document.getElementById('sourceImage');
  context.drawImage(image, 0, 0);
}
```

8. Our first image manipulation function is called `grayscaleImage`. Let's include this now and within it we'll loop through the pixel data of our image, which we can retrieve from the `canvas` element using the `getImageData` method, as shown in the following code block:

```
function grayscaleImage() {
  var imageData = context.getImageData(0, 0, 300, 300);
  for (var i = 0, n = imageData.data.length; i < n; i += 4) {
    var grayscale = imageData.data[i] * .3 +
    imageData.data[i+1] * .59 + imageData.data[i+2] * .11;
   imageData.data[i] = grayscale;
   imageData.data[i+1] = grayscale;
   imageData.data[i+2] = grayscale;
  }
  context.putImageData(imageData, 0, 0);
}
```

9. Our second manipulation function is called `processSepia`. Once again, we obtain the image data from our `canvas` element and loop through each pixel applying the changes as we go.

```
function processSepia() {
var imageData =
  context.getImageData(0,0,canvas.width,canvas.height);
  for (var i=0; i < imageData.data.length; i+=4) {
    imageData.data[i] = r[imageData.data[i]];
    imageData.data[i+1] = g[imageData.data[i+1]];
    imageData.data[i+2] = b[imageData.data[i+2]];
    if (noise > 0) {
      var noise = Math.round(noise - Math.random() * noise);
      for(var j=0; j<3; j++){
        var iPN = noise + imageData.data[i+j];
        imageData.data[i+j] = (iPN > 255) ? 255 : iPN;
      }
   }
  }
  context.putImageData(imageData, 0, 0);
};
```

10. When we run the application on the device, after selecting a button to change our default image the output would look something like the following screenshot:

How it works...

When we start to process a change to the canvas image, we first obtain the data using the `getImageData` method available through the `canvas` context. We can easily access the information for each pixel within the returned image object and its `data` attribute.

With the `data` tag in an array, we can loop over each pixel object, and then over each value within each pixel object.

Pixels contain four values: the red, green, blue, and alpha channels respectively

By looping over each specific color channel in each pixel, we can alter the values, thereby changing the image. We can then set the revised image as the source in our canvas by using the `putImageData` method to set it back in to the context of our `canvas`.

There's more...

Although this recipe does not involve any PhoneGap specific code with the exception of the `onDeviceReady` method, it was included here for the following three reasons:

- As an example to show you how you might like to work with images captured using the PhoneGap API
- To remind you of or introduce you to the power of HTML5 elements and how we can work with the `canvas` tag
- Because it's pretty cool