Chapter No. 2
"Pages and Dialogs"
In this package, you will find:
A Biography of the author of the book
A preview chapter from the book, Chapter NO.2 "Pages and Dialogs"
A synopsis of the book’s content
Information on where to buy this book

About the Author

**Chetan K Jain** loves to code, and has been writing code for over 16 years now. He is a Senior Architect, and has worked on mobile technologies for over 4 years for Nokia. Since then, he has moved on to work as a freelance consultant.

Chetan has significant experience in writing cross-platform mobile apps using jQuery Mobile, HTML5, CSS3, JavaScript, Nodejs, and has also worked extensively on Qt. Prior to mobile technologies, he has worked with Java technology and was also certified as an MCSD and MCSE in his early days.

Chetan is an active contributor to open source development, and tries to help jQuery Mobile development whenever he can. He regularly participates in developer forums, and was earlier a top-ranked member and a "mad scientist" in the Nokia Qt Developer Forum.

For More Information:
Chetan was born in Bangalore and lives there with his wife Shwetha and son Tanmay. The only time he ventured to live away from Bangalore was when he worked for over 4 years in the USA. His adventure didn't end there. He did a solo US cross country drive in his two door coupe from the east coast to the west and back, driving alone for over 8000 plus miles in 16 days to experience and live life as he calls it. His travelogue can be found at http://adventure.chetankjain.net.

Chetan has contributed and published over 15 books to Gutenberg as a volunteer. He has also published four books on Jaina Literature and History written by his mother Saraswathamma. He is a voracious reader, and his other interests include music, movies, and travelling. Photography is his favorite hobby, and his clicks can be found at http://www.facebook.com/chetankjainphotos/photos_stream.

Chetan can be reached at chetankjain@gmail.com. He also blogs at http://dev.chetankjain.net.

For More Information:
jQuery Mobile Cookbook

jQuery Mobile is an award winning, HTML5/CSS3-based open source, cross-platform UI framework. It offers a very cool and highly customizable UI. It is built on the popular jQuery library and uses declarative coding, making it easy to use and learn. It is the market leader today, considering the numerous browsers and platforms that it supports.

jQuery Mobile Cookbook presents over eighty recipes written in a simple and easy manner. You can quickly learn and start writing the code immediately. Advanced topics, such as using scripts to manipulate, customize, and extend the framework, are also covered. These tips address your common everyday problems. The book is very handy for both beginner and experienced jQuery Mobile developers.

You start by developing simple apps using various controls and learn to customize them. Later, you explore using advanced aspects, such as configurations, events, and methods.

Develop single and multi-page applications. Use caching to boost performance. Use custom transitions, icon sprites, styles, and themes. Learn advanced features, such as configurations, events, and methods. Explore the new features and semantics of HTML5 using it with jQuery Mobile.

jQuery Mobile Cookbook is an easy read, and is packed with practical tips and screenshots.

What This Book Covers

Chapter 1, Get Rolling, begins with a brief introduction on what the jQuery Mobile framework is and what it can do for you. You will get to write your first jQuery Mobile cross-platform app here. You will also see how to use the online JSBin tool to develop and test your apps.

Chapter 2, Pages and Dialogs, here you will learn how to compare and use single page and multi-page template applications. You will learn various performance-enhancing techniques, such as prefetching and using the DOM cache to improve your page loading speed. You will create new custom transitions using JavaScript and CSS, and also learn to use page redirection for a login page. You will also create a custom styled dialog, and use the HTML5 History API to create your own custom pop up.

Chapter 3, Toolbars, here you will learn how to use fixed and full screen toolbars and how to persist your navigation links across pages. You will see how you can create and add custom round buttons, images, and a custom back button to the header, and a grid layout to the footer.

For More Information:
Chapter 4, *Buttons and Content Formatting*, here you will use JavaScript to dynamically create a button and assign an action to it. Then, you will learn how to use a custom icon, add a custom icon sprite, and finally replace the existing icon sprite provided by the jQuery Mobile framework. You will learn how to create nested accordions (collapsible sets), how to create a custom layout grid, and finally see how to format and display XML and JSON content in your app.

Chapter 5, *Forms*, shows you how to natively style forms, disable text controls, and group radio buttons into a multi-row grid. You will learn to customize a checkbox group, auto initialize select menus, and create dynamic flip switch and slider controls. You will also see how to validate and submit a form to a server using POST, and also how to fetch data using GET. Finally, you will learn how to create an accessible form.

Chapter 6, *List Views*, here you will learn how to use various list types and also customize them. You will use an inset list, custom number a list, and then create a read-only list. You will see how to format list content, use a split button, and an image icon list. You will also create a custom search filter for your list, and finally see how you can use JavaScript to modify a list.

Chapter 7, *Configurations*, shows you how to tweak, configure, and customize the various options and settings provided by the jQuery mobile framework. Configuring the active classes, enabling Ajax, auto initializing pages, configuring default transitions, customizing error and page loading messages, and using your own custom namespace are all covered along with a few more advanced configuration options.

Chapter 8, *Events*, shows you how to use the various events available in the framework. You will learn to use the orientation, scroll, touch, virtual mouse, and layout events along with the page initialization, page load, page change, and page remove events. You will also see how to use the page transition and animation events.

For More Information:
Chapter 9, *Methods and Utilities*, here you will see how to use the methods and utilities provided in the framework. The chapter runs through the methods provided by the framework and lists working recipes for each of these. You will see how to load a page, change a page, and also how to do silent scrolling.

Chapter 10, *The Theme Framework*, here you will learn how to theme a nested list, style button corners, and use custom backgrounds and fonts. You will explore how to override the global active state and override an existing swatch. Finally, you will use the ThemeRoller web tool to create and use your own swatch.

Chapter 11, *HTML5 and jQuery Mobile*, here you will see how to use various HTML5 features in your jQuery mobile app. You will explore some new HTML5 semantics, use the Application Cache to take your app offline, use Web Workers to see how asynchronous operations are done, and you will use web storage to store data using local and session storage. Then you will see how to draw in 2D using the Canvas, use SVG image and apply a Gaussian blur filter on it, track your device location using the Geolocation API, and finally see how to use audio and video in your app.

For More Information:

In this chapter, we will cover:

- Writing a single-page template application
- Writing a multi-page template application
- Prefetching pages for faster navigation
- Using the DOM cache to improve performance
- Custom styling a dialog
- Using CSS to create a bouncing page transition
- Using JS to create a slide and fade page transition
- Using `data-url` to handle a login page navigation
- Using the History API to create a custom error pop up

### Introduction

**Page** is the basic jQuery Mobile object written within a `<div data-role="page">` container that gets displayed on the screen. It can contain the header, the page content, and the footer. You can embed various HTML5 controls and widgets within a page. The jQuery Mobile framework automatically enhances and displays all these controls, making them tap-friendly (finger-friendly). Your application can have a series of individual HTML files each representing a single page, or it can have one single HTML file containing multiple page `div` containers within it. You can provide links to open other pages within a page, and when the user clicks on a link, the new page opens using Ajax with CSS3 animation. The current page is then hidden from view.

**Dialog** is a page having the `data-role="dialog"` attribute. You can also load a page as a dialog by adding the `data-rel="dialog"` attribute to the page link. The dialog is styled differently from a page, and it appears in the middle of the screen above the page. The dialog also provides a close button in its header.

For More Information:
Writing a single-page template application

In a **single-page template** application, each page of the application will have its own HTML file. A page is wrapped within a page container as `<div data-role="page">`. When you launch the app, the jQuery Mobile framework will load the first page of the app (or the main page) into the DOM, whose reference is held all through the app cycle. The main page just gets hidden when the user navigates to another page, which now is marked as an active page. Except for the main page, all other pages get removed from the DOM when the user navigates away from them. Navigation between the pages is specified using anchor links. The anchor links are decorated as buttons using the `data-role="button"` attribute. On clicking any link, navigation occurs with some cool CSS3 transitions, and the new page is pulled in via Ajax.

This recipe shows you how to create a single-page template application and navigate between the pages of the app.

**Getting ready**

Copy the full code of this recipe from the code/02/single-page sources folder. You can launch this code using the URL [http://localhost:8080/02/single-page/main.html](http://localhost:8080/02/single-page/main.html).

**How to do it...**

Carry out the following steps:

1. Create `main.html`, and add a page container with the header, footer, and page content to it. Add a link to open `page2.html`:

```html
<div id="main" data-role="page">
    <div data-role="header">
        <h1>Header of main.html</h1>
    </div>
    <div data-role="content">
        <a href="page2.html" data-role="button">
            Go to Page 2
        </a>
    </div>
    <div data-role="footer">
        <h4>Footer of main.html</h4>
    </div>
</div>
```

For More Information:

2. Since this is a single-page template app, add each page to its own HTML file. Next, create page2.html and add the second page of the app to it. Add a link to go back to main.html:

```
<div id="page2" data-role="page">
  <div data-role="header">
    <h1>Header of page2.html</h1>
  </div>
  <div data-role="content">
    <a href="#" data-role="button" data-rel="back" data-theme="b">Go Back</a>
  </div>
  <div data-role="footer">
    <h4>Footer of page2.html</h4>
  </div>
</div>
```

**How it works...**

Create main.html, and add a page to it using the `<div>` page container with the `data-role="page"` attribute specified. Add the header, footer, and page content, as shown in the code. Now, add an anchor link to the page content to open the second page, page2.html. You can style this link as a button by using the `data-role="button"` attribute.

Next, create page2.html and add a page to it using the `<div>` page container with the `data-role="page"` attribute specified. Add the header, footer, and page content to it, as shown in the code listing. Here, in the page content, add an anchor link to go back to main.html. Also, set the `data-role="button"` attribute to style this link as a button.

Now, when you launch the app, the main.html page is first loaded into the DOM. This page stays in the DOM throughout the life cycle of the app. The following screenshot is displayed:

![Header of main.html](image1)

![Go to Page 2](image2)

![Footer of main.html](image3)

For More Information:  
When you click on the button to open page2.html, the main page is hidden from view, and page2.html is displayed and made active, as shown in the following screenshot:

![Header of page2.html](image1)

Now, click on the link to go back to main.html. The browser opens the main.html page again and hides page2.html.

In page2.html, the anchor button has a data-rel="back" attribute. This indicates that the previous page in the browser history should be loaded. The href link is ignored and so you can set it to #.

### Setting the title for a single-page template app

Use the `<title>` tag to set the page title for each page in a single-page app. This ensures that relevant titles are displayed as you navigate through the various pages in your app.

## There's more...

It is recommended that most applications use single-page templates, for the following reasons:

- Pages are lighter, cleaner, and more modular, and thus easier to maintain.
- The DOM size is relatively smaller.
- Pages work well on multiple platforms and environments. They work even where JavaScript is not supported. You can target more devices this way.

On the flip side:

- It consumes more bandwidth as each page visit generates a new request.
- Opening a previously loaded page again will generate a fresh request.
- First load is faster, but every subsequent page has to be fetched.

To conclude, single-page template apps are more suited for larger applications and in situations where you want to target as many platforms as possible.

---

For More Information:

Turning off Ajax navigation

In this recipe, in #page2, the href value is set to #. If you set the href value to the absolute or relative URL of the page, that is href="main.html", then Ajax navigation will still work. To prevent pages being loaded via Ajax, add the data-ajax="false" attribute to the link. The framework will not use the custom CSS3 transitions when Ajax is turned off.

```html
<a href="page2.html" data-role="button" data-ajax="false">text</a>
```

**Using URL instead of data-rel="back"**

It is always better to use URLs in href of the anchor link while navigating in single page apps. This way, Ajax navigation would work where Ajax is supported. In C grade browsers, where Ajax is not supported, the app would still continue to work, since it uses href for navigation. In such browsers, if your app relies only on the data-rel="back" attribute, and does not use href, then page navigation would break down.

**Using data-rel and data-direction**

When you add both the href and data-rel="back" attributes to an anchor link, the href attribute is ignored by the framework. The page will only consider the data-rel attribute and navigate "back"; that is, it will navigate to the page present as the previous entry in the browser history stack. If you specify the data-direction="reverse" attribute, the framework will reverse the direction of the most recent page transition used. The data-direction attribute does not depend on the data-rel attribute, and can be used independently in any transition.

```html
<a href="page2.html" data-role="button"
   data-direction="reverse">text</a>
```

**Page container is optional**

Specifying the `<div data-role="page">` page container is optional in a single-page template application. The page contents are automatically wrapped with a page container by the jQuery Mobile framework.

Always use a div page container to wrap your page. It is easier to read and maintain the code. It also allows you to add page-specific data attributes such as data-theme to your page.

**See also**

- The Writing a multi-page template application, Prefetching pages for faster navigation, and Using the DOM Cache to improve performance recipes
- The Writing your first jQuery Mobile application recipe in Chapter 1, Introduction

**For More Information:**

Writing a multi-page template application

In a multi-page template application, the HTML file will have multiple pages in it. Each page is wrapped within a page container as `<div data-role="page">`. The page ID is used to identify the pages for linking or invoking any actions on them. The page ID must be unique within your app. When you launch the app, the jQuery Mobile framework loads all the available pages into the DOM and displays the first page it finds in the HTML. Navigation between the pages is specified by using anchor links, and you can decorate these links as buttons by using the `data-role="button"` attribute. On clicking any link, navigation occurs with some cool CSS3 transitions, and the new page is pulled in via Ajax. This recipe shows you how to create a multi-page template application and navigate between the multiple pages it contains.

Getting ready

Copy the full code of this recipe from the code/02/multi-page sources folder. You can launch this code using the URL http://localhost:8080/02/multi-page/main.html.

How to do it...

Carry out the following steps:

1. Create `main.html`, and add the #main page to it. Define the header, page content, and footer, as shown in the following code snippet. Add a link to open the #page2 page in the page content:

```html
<div id="main" data-role="page">
  <div data-role="header">
    <h1>Header of #main</h1>
  </div>
  <div data-role="content">
    <a href="#page2" data-role="button">Go to Page 2</a>
  </div>
  <div data-role="footer">
    <h4>Footer of #main Page</h4>
  </div>
</div>
```

2. Next, in `main.html`, add the second #page2 page in its own page div container, as shown in the following code snippet. Add the header, page content, and footer to this page. Finally, add a link to go back to the #main page in its page content:

```html
<div id="page2" data-role="page" data-title="Multi-Page Template">
  <div data-role="header">
    <h1>Header of #page2</h1>
  </div>
</div>
```

For More Information:

<div data-role="content">
    <a href="#" data-role="button" data-rel="back" data-theme="b">Go Back</a>
</div>
<div data-role="footer">
    <h4>Footer of #page2</h4>
</div>

Downloading the example code
You can download the example code files for all Packt books you have purchased from your account at http://www.PacktPub.com. If you purchased this book elsewhere, you can visit http://www.PacktPub.com/support and register to have the files e-mailed directly to you.

How it works...
Create main.html, and add two pages, #main and #page2, to it. First, add the #main page using the <div> page container with the data-role="page" attribute specified. Add the header, footer, and page content, as shown in the code. Now, add an anchor link to the page content to open the second page, #page2. You can style this link as a button by using the data-role="button" attribute.

Next, add the #page2 page using the <div> page container, with the data-role="page" attribute specified. Add the header, footer, and page content to it as shown in the code listing. Here, in the page content, add the anchor link to go back to the #main page. Set the data-role="button" attribute to style it as a button. Also, add the data-rel="back" attribute to it. This indicates to the jQuery Mobile framework that this link should open the previous page available in the browser history.

Now, when you launch the app, all the pages are loaded into the DOM and they stay in the DOM throughout the life cycle of the app. The framework opens the first page it finds. So, #main is displayed with a button to open #page2, as follows:

<table>
<thead>
<tr>
<th>Header of #main page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Go to Page 2</td>
</tr>
<tr>
<td>Footer of #main page</td>
</tr>
</tbody>
</table>

For More Information:
When you click on the button to open the second page, the #main page is hidden from view, and the #page2 page is displayed and made active, as follows:

Finally, click on the link to go back to the #main page. Since data-rel="back" was used, the browser opens the #main page again and hides #page2.

### Setting the title for a multi-page template app
Use the <title> tag to set the page title for the first or the main page of the multi-page template app. Use the data-title attribute to set the title of all the other pages. This will ensure that the correct titles are shown for each page.

### There's more...
It is recommended that the following factors be considered before using a multi-page template to build your app:

- A multi-page template application is heavier due to the large DOM size.
- The DOM size is relatively larger and heavier because all the pages are loaded into it upfront.
- The application needs JavaScript support. This limits your choice of target platforms, and you might have to ignore many popular legacy platforms. But this exclusion list is getting thinner by the day as the older phones/platforms get phased out.

On the plus side:

- Only the first page load is slower, but subsequent page navigations are fast.
- All the pages are pre-loaded in the DOM, so no new requests (to the server) are required for subsequent page navigations. This means lesser bandwidth.

To conclude, multi-page template apps are more suited for relatively smaller applications and situations where you know the capabilities of your target platforms, including JavaScript support.
The updated list of browsers and platforms supported by jQuery Mobile is available at http://www.jquerymobile.com/gbs. It also details the grade of support provided on these platforms.

Using transitions
The data-transition attribute can be used to specify various transitions that are available by default with jQuery Mobile. The following code opens #page2 using a flip transition:

```
<a href="#page2" data-transition="flip" data-role="button">text</a>
```

Turning off Ajax navigation
If you pass the data-ajax="false" attribute while loading a page in a multi-template app, it does not completely stop Ajax navigation. The default fade transition will then be used to load the pages regardless of the transition specified in the data-transition attribute.

```
<a href="#page2" data-ajax="false" data-role="button">text</a>
```

Page container is mandatory
Specifying the `<div data-role="page">` page container is mandatory for all the pages within a multi-page template application. Use the page container for all your apps and all your pages regardless of whether they use the single-page or multi-page templates.

See also
- The Writing a single-page template application, Prefetching pages for faster navigation, and Using the DOM Cache to improve performance recipes
- The Writing your first jQuery Mobile application recipe in Chapter 1, Introduction

Prefetching pages for faster navigation
Using a single-page template for your mobile app makes your mobile app faster and lighter. But you have to fetch each page during navigation. You can see the ui-loader spinning icon every time a page loads. This problem does not happen with a multi-page template application, as all the pages are already preloaded into the DOM. By using the prefetch feature, a single-page template application can be made to mimic the multi-page template application.

A page marked for prefetch is loaded in the background and is immediately available when the user tries to open it. You can prefetch pages in two ways. The first is by just adding the data-prefetch attribute to the anchor link. The second way is by using JavaScript to call the loadPage() method. This recipe shows you how to improve page loading speed by prefetching pages in your jQuery Mobile app.

For More Information:
Getting ready

Copy the full code of this recipe from the code/02/prefetch sources folder. You can launch this code using the URL http://localhost:8080/02/prefetch/main.html.

How to do it...

The steps to be followed are:

1. Create main.html and add two links to it. The first link points to prefetch.html and the second link to prefetch-JS.html. After the main.html file is loaded, the linked pages in it can be prefetched in the background using the data-prefetch attribute on the first link, as shown in the following code snippet:

   ```html
   <div id="main" data-role="page">
   <div data-role="header">
   <h1>Header of Main Page</h1>
   </div>
   <div data-role="content">
   <a href="prefetch.html" data-role="button" data-prefetch>Prefetch Page</a>
   <a href="prefetch-JS.html" data-role="button">Prefetch Page using JS</a>
   </div>
   <div data-role="footer">
   <h4>Footer of Main Page</h4>
   </div>
   </div>
   ```

2. Next, add the JavaScript given in the following code snippet to the <head> section of main.html. Here, use the loadPage() method to load the prefetch-JS.html file in the background, into the DOM:

   ```javascript
   $("#main").live("pageshow", function(event, data) {
   $.mobile.loadPage( "prefetch-JS.html",
   { showLoadMsg: false });
   });
   ```

3. Now, create the prefetch.html file as shown in the following code snippet. This is a regular page which is prefetched in the main.html page (in step 1), using the data-prefetch attribute. Also add a link to navigate back to main.html:

   ```html
   <div id="prefetch" data-role="page">
   <div data-role="header">
   <h1>Header of Prefetched Page</h1>
   </div>
   ```

For More Information:

4. You will see that in step 2, the `prefetchJS.html` was prefetched using JavaScript. Now, create `prefetchJS.html`, as shown in the following code snippet, and add a link to navigate back to `main.html`:

```html
<div id="jsprefetch" data-role="page">
  <div data-role="header">
    <h1>Header of JS Prefetched Page</h1>
  </div>
  <div data-role="content">
    <a href="#" data-role="button" data-rel="back" data-theme="b">Go Back</a>
  </div>
  <div data-role="footer">
    <h4>Footer of JS Prefetched Page</h4>
  </div>
</div>
```

How it works...

Create `main.html`, and add two links to it. Point the first link to `prefetch.html` and set the `data-prefetch` attribute to this link. This page is now automatically fetched in the background and is immediately available for opening when `main.html` is loaded.

Point the second link to the `prefetch-JS.html` file. To prefetch this page using JavaScript, add an event handler for the `pageshow` event of `#main`. In this callback function, invoke the `loadPage()` method to fetch the `prefetch-JS.html` file. Also set the `showLoadMsg` option as `false`, to prevent the spinning page `ui-loader` message from being shown.

Next, create the two HTML files as shown in the code. Add the link to navigate back to `main.html` in both the pages.

For More Information:

Now, when you launch the app, both the HTML files are prefetched. You can observe this prefetch behavior using your browser’s code inspector, as shown in the following screenshot:

This screenshot shows the code inspector in the Google Chrome browser right after loading the main.html page. We can see that both the #prefetch and #jsprefetch pages are already prefetched and available in the DOM. Now, navigating to these prefetched pages is almost immediate, and the spinning ui-loader icon animation does not show up. This makes your app much faster and gives a better user experience to the user. If prefetch was not used, the page would have loaded only when you navigated to it.

Using the data-prefetch attribute is the easier way to prefetch pages, as you don’t have to write any further code with it. But prefetching a page with JavaScript using loadPage() allows you to provide more options to the loadPage() method and have a better control over the behavior of your page load. You can also build in conditional prefetch with this approach.

**There’s more…**

Avoid using prefetch on too many pages, as all the pages have to be fetched and stored in the DOM. This means more memory utilization, and memory is a scarce resource on mobile devices. This will slow down your app. More pages prefetched also means more bandwidth utilized. So use it judiciously.

**When prefetch is not completed**

If a page hasn’t been fully prefetched and you try to navigate to that page, then the ui-loader spinner comes up and the page is shown only after the page is completely fetched. This could occur on slower connections.

For More Information:

Prefetched pages are not permanently cached

When a page is prefetched, it is available in the DOM. If you navigate to this page and then navigate away, the page is automatically removed from the DOM. So if it is a frequently visited page, you have to add it to the DOM cache instead.

See also

- The Using the DOM cache to improve performance recipe
- The Using loadPage() to load a page recipe in Chapter 9, Methods and Utilities

Using the DOM cache to improve performance

During page navigation in a single-page template application, each new page is fetched and stored in the DOM. The page remains in the DOM and is removed once you navigate away from the page. Only the main or the first page of the app always remains in the DOM.

As seen in the previous recipe, prefetching commonly-used pages could help in improving performance to some extent. But when you visit a prefetched page and navigate away from it, the page gets removed from the cache. So the problem of multiple fetching of frequently visited pages is not fully solved.

With DOM caching, specific pages are marked to be cached in the DOM. These pages, once loaded, remain in the DOM all through the life cycle of the app. You can use the DOM cache in two ways. The first is by adding the data-dom-cache attribute to the page container of the page that is to be cached. The second way is by using JavaScript. This recipe shows you how to improve the performance of your app by using the DOM cache.

Getting ready

Copy the full code of this recipe from the code/02/dom-cache sources folder. You can launch this code using the URL http://localhost:8080/02/dom-cache/main.html.

How to do it...

The steps to be followed are:

1. Create the main.html file with links to navigate to the the two pages, cached.html and cachedJS.html. Both these pages, in turn, specify that they should be cached in the DOM:

```html
<div id="main" data-role="page">
  <div data-role="header">
    <h1>Header of Main Page</h1>
  </div>
```

For More Information:

2. Create the cached.html page and set the data-dom-cache attribute of its page container. Also add a button to go back to the main.html page:

```html
<div id="cached" data-role="page" data-dom-cache="true">
  <div data-role="header">
    <h1>Header of Cached Page</h1>
  </div>
  <div data-role="content">
    <a href="#" data-role="button" data-rel="back">
      Go Back
    </a>
  </div>
  <div data-role="footer">
    <h4>Footer of Cached Page</h4>
  </div>
</div>
```

3. Finally, create the cached-JS.html file, and cache it by using JavaScript added to the page div container, as shown in the following code snippet. Add a button to navigate back to main.html:

```html
<div id="jscached" data-role="page">
  <script>
    $("#jscached").page({ domCache: true });
  </script>
  <div data-role="header">
    <h1>Header of JS Cached Page</h1>
  </div>
  <div data-role="content">
    <a href="#" data-role="button" data-rel="back">
      Go Back
    </a>
  </div>
</div>
```

For More Information:
How it works...

Create `main.html` and add two links to open the `cached.html` and `cached-JS.html` files. Next, create the `cached.html` file with a link to go back to `main.html`. Here, set the `data-dom-cache="true"` attribute to the page container. This indicates that the page must be cached in the DOM once it is loaded.

Now create the `cached-JS.html` file with a link to return to `main.html`. Here, add the given script to the page `div` container. In the script, set the `domCache` option on the page to `true`. Now, when this page is loaded, it gets cached in the DOM.

Launch the app and navigate between the pages. During page navigation, each new page is fetched and stored in the DOM. You can observe the DOM cache behavior using your browser’s code inspector. The following image shows the Chrome code inspector snapshot after both the pages were visited and cached in the DOM. The current active page is shown as `#main`; this is indicated by the `ui-page-active` class added to its page’s `div` container. The other two pages are cached and are also seen available in the DOM.

For More Information:

Adding scripts to the page div and not the <head> element

When using Ajax navigation, the <head> section is processed only on the first page or the main page of your app. The <head> element of each of the remaining pages is ignored and only their page's div containers are processed. Thus, to ensure that your script is executed in these pages, you have to include the <script> tag within the page's div container.

There's more...

If you want to cache all pages ever visited in your app, it becomes cumbersome to add the caching option in each of these pages. There is a way to do this globally using JavaScript. Add the following script to the <head> section of your main page. Now, every page visited automatically gets cached in the DOM.

```
<script>
    $.mobile.page.prototype.options.domCache = true;
</script>
```

DOM caching can slow down your app

Having a large number of pages cached in the DOM could make your app very heavy and slow it down. In such situations, you will have to write extra code to manage the cached pages in the DOM, and perform any clean ups that are required. So, use DOM caching on selected frequently accessed pages only.

See also

- The Prefetching pages for faster navigation recipe

Custom styling a dialog

You can style a page as a dialog by using the data-role="dialog" attribute on the page container. You can also specify the data-rel="dialog" attribute in the anchor link used to open the page. The page now gets styled as a dialog, and opens with a pop transition. When you add a header to the dialog, a close icon is created on the header, by default, in the left side of the header. In some applications/platforms, you might want to position this close button on the right side of the header. There is no ready option available to change this icon's position. This recipe shows you how to build a dialog with a custom styled header to position the close button at the right side of the header.
Getting ready

Copy the full code of this recipe from the code/02/custom-dialog sources folder. You can launch this code using the URL http://localhost:8080/02/custom-dialog/main.html.

How to do it...

The steps to be followed are:

1. Create main.html with the #main page. Add a link here to open the #customdialog page as a dialog using the data-rel="dialog" attribute:

```html
<div id="main" data-role="page">
  <div data-role="header">
    <h1>Header of Main Page</h1>
  </div>
  <div data-role="content">
    <a href="#customdialog" data-role="button" data-rel="dialog">Open Custom Dialog</a>
  </div>
  <div data-role="footer">
    <h4>Footer of Main Page</h4>
  </div>
</div>
```

2. Create the #customdialog page in main.html, and add the custom header to the dialog that positions the close button on the right side of the header. The default header enhancement is prevented in this code:

```html
<div id="customdialog" data-role="page">
  <div class="ui-corner-top ui-overlay-shadow ui-header ui-bar-a" role="banner">
    <a href="#main" data-icon="delete" data-iconpos="notext" class="ui-btn-right ui-btn ui-btn-icon-notext ui-btn-corner-all ui-shadow ui-btn-up-a" title="Close" data-theme="a" data-transition="pop" data-direction="reverse">
      <span class="ui-btn-inner ui-btn-corner-all">
        <span class="ui-btn-text">Close</span>
        <span class="ui-icon ui-icon-delete ui-icon-shadow"></span>
      </span>
    </a>
    <h1 class="ui-title" tabindex="0" role="heading" aria-level="1">Custom Dialog</h1>
  </div>
</div>
```

For More Information:

3. Finally, add the page content with a link to go back to the #main page:
   <div data-role="content">
      <a href="#" data-role="button" data-rel="back" data-theme="b">Go Back</a>
   </div>
   <div data-role="footer">
      <h4>Footer of Dialog</h4>
   </div>
</div>

### How it works...

Create main.html with two pages, #main and #customdialog, in it. Add a link in the #main page to open the #customdialog page as a dialog, by setting the data-rel="dialog" attribute. Next, create the #customdialog page and add a button to go back to the #main page. Now, in the header of #customdialog, do not use the data-role="header" attribute. This will prevent the dialog header from being enhanced with the default style. The close icon will not be placed at the left side of the header now. You can now add your custom header and set custom styles to it, as given in the code listing earlier. Launch the app and open the dialog, you will see the dialog pop up. This dialog now has a custom styled header with the close icon on the right side of the header, as shown in the following screenshot:

![Custom Dialog](image)

To understand how the custom style was arrived at, first create a page that opens a regular dialog. Using the code inspector of your browser and observe the code enhancements done by the jQuery Mobile framework to the header of the dialog. Copy this generated code "as is" into your custom dialog code. Then you have to make the changes mentioned in the following sections.

The first change is to fix the close icon's position. You will see that the close action is performed with the help of an anchor link that has been added into the header code. Here, replace the ui-btn-left class with the ui-btn-right class. This will position the icon to the right in the header. The jquery.mobile.css file already has these class definitions in it.

---

For More Information:

With this change, the close icon now appears at both the left and the right positions in the header. This is because the header still has the `data-role="header"` attribute. This makes the framework enhance the entire header and automatically add the close icon on the left side. But, since you have already added all these generated classes manually, you can now safely remove the `data-role="header"` attribute from your code. Retain all the other code and classes that you have added. Now, when you launch the code, you will only see a single close icon positioned at the right side of your header.

**There's more...**

This technique is a very important one. It can be used to customize how your jQuery Mobile apps should look and feel. The framework provides many basic options, elements, and attributes that you can add to your apps. The framework then enhances these by adding more markup code and styles internally, making it look good in your browser. This enhanced code is not visible in the View Source option of your browser. But, with a code inspector or debugging tool, you can view the enhanced code, copy it to your HTML files, tweak it, and get the result you want. The following screenshot shows the code inspector view for the custom dialog header created using this recipe:

![Custom dialog header code inspector view](image)

### Customizing CSS

The dialog page can further be enhanced by introducing your own styles in a custom CSS file. Check for all classes that have `ui-dialog` in the `jquery.mobile.css` file. Copy the styles that you want to tweak into your custom CSS and set appropriate new values. The following line of code shows a sample change where the top margin of the dialog is set to `-12px` instead of the default `-15px`:

```css
.ui-dialog { margin-top: -12px; }
```

### See also

- The Adding a customized round button to the header recipe in Chapter 3, Toolbars

For More Information:

Using CSS to create a bouncing page transition

As you navigate between the pages of your app, the jQuery Mobile framework uses CSS3 animations to show some cool transition effects. The fade transition is used by default for pages, and the pop transition is used for dialogs. You can navigate into a page with a particular transition, and when you navigate out of the page, you can reverse the direction of the transition. jQuery Mobile comes with a default set of 10 transitions as of v1.1.1. The jQuery Mobile online docs has a nice online demo showing all the available transitions. But that's not all; you can use CSS to create your own custom transitions and use them in your app. This recipe shows you how to use CSS and create a bouncing page effect during page transitions.

Getting ready

Copy the full code of this recipe from the code/02/custom-css-transition sources folder. You can launch this code using the URL http://localhost:8080/02/custom-css-transition/main.html.

How to do it...

The steps to be followed are:

1. Create the customtransition.css file, and define the bounceup custom transition as shown in the following code snippet. Animate the Y position property of the page in the CSS:

```
.bounceup.in, .bounceup.in.reverse {
  -webkit-transform: translateY(0);
  -webkit-animation-name: bounceupin;
  -webkit-animation-duration: 1s;
  -webkit-animation-timing: cubic-bezier(0.1, 0.2, 0.8, 0.9);
}

@-webkit-keyframes bounceupin {
  0% { -webkit-transform: translateY(100%); }
  90% { -webkit-transform: translateY(-10%); }
  100% { -webkit-transform: translateY(0); }
}
```

2. Define the reverse animation next:

```
.bounceup.out, .bounceup.out.reverse {
  -webkit-transform: translateY(100%);
  -webkit-animation-name: bounceupout;
  -webkit-animation-duration: 1s;
  -webkit-animation-timing: cubic-bezier(0.1, 0.2, 0.8, 0.9);
}
```
3. Create main.html and include the reference to the customtransition.css stylesheet in its <head> section, as follows:

```html
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet" href="http://code.jquery.com/mobile/1.1.1/jquery.mobile-1.1.1.min.css" />
<link rel="stylesheet" href="customtransition.css" />
<script src="http://code.jquery.com/jquery-1.7.1.min.js"></script>
<script src="http://code.jquery.com/mobile/1.1.1/jquery.mobile-1.1.1.min.js"></script>
```

4. Create the #main page with a link to open #page2. Set the bounceup custom transition defined earlier to the data-transition attribute:

```html
<div id="main" data-role="page">
  <div data-role="header">
    <h1>Header of Main Page</h1>
  </div>
  <div data-role="content">
    <a href="#page2" data-role="button" data-transition="bounceup">Go to Page 2</a>
  </div>
  <div data-role="footer">
    <h4>Footer of Main Page</h4>
  </div>
</div>
```

5. Finally, create the #page2 page with a link to go back to the #main page:

```html
<div id="page2" data-role="page" data-title="Custom Transition using CSS">
  <div data-role="header">
    <h1>Header of Page 2</h1>
  </div>
  <div data-role="content">
    <a href="#" data-role="button" data-rel="back" data-theme="b">Go Back</a>
  </div>
</div>
```

For More Information:
How it works...

Create the customtransition.css file and define the custom bounceup transition. First, define the .bounceup.in and .bounceup.in.reverse classes, with both having the same values. This will make both the transitioning into a new page and out of it (reverse) look similar. In the class, set the Y co-ordinate or the vertical position of the new page on the screen using the translateY property. Animate this property over the given duration of 1 second, using a Cubic Bezier animation curve. Next, define the keyframes for the animation of the Y co-ordinate (this is specified using the bounceupin animation name). The keyframes define the values of Y at various times within the animation.

You can use a simple trick to get the bounce effect that is used in this animation. Set the value of Y to beyond the screen at 90 percent duration, and then set it to the edge of the screen at 100 percent duration, or at the completion of the animation. This gives it a neat bouncing effect as the new page animates into the screen, extends out of the screen for a short duration, and comes back to the correct position. Similarly, define the .bounceup.out and .bounceup.out.reverse animations for the current page while it navigates out of the screen, as shown in the code.

Now, create main.html and include the CSS file in its <head> section after including the jquery.mobile.css file. Create the #main page, and add a link to open the #page2 page with the custom transition using the data-transition="bounceup" attribute. Finally, create the #page2 page with a link to go back to the #main page. Now when you launch the app and click on the buttons, the page navigation will occur, using a nice custom bouncing animation.

During a page transition, there is a from and a to page. jQuery Mobile applies the out class style on the from page (current page) and the in class style on the to page (new page). If the reverse transition is to be supported, the word reverse is suffixed to the in and out classes, as shown in the CSS file. Using these styles, jQuery Mobile will apply the right transition effects on the pages. You can further tweak the code in this recipe, and explore further with CSS animations to create more page animations. You can get as creative as you want!

For More Information:
There's more...

The CSS styles are listed in this recipe to support only the web kit browsers (Chrome and Safari). You can explore this further and try to make it work on other browsers, such as IE, Firefox, or Opera. You will have to add vendor-specific prefixes to the CSS properties. Also, the browser should be capable of supporting the CSS property used. The vendor prefixes required for the popular browsers are as follows:

- **Chrome and Safari**: -webkit
- **Opera**: -o
- **Firefox**: -moz
- **IE**: -ms

**Adding vendor prefixes to the customtransition.css file**

To incorporate support for other browsers, you will have to extend the customtransition.css file provided in this recipe. You can do this by adding vendor prefixes for the properties, as follows:

```css
.bounceup.in, .bounceup.in.reverse {
  -webkit-transform: translateY(0);  
  -moz-transform: translateY(0);     
  -ms-transform: translate(0)       
  -o-transform: translate(0)       
  transform: translate(0)          
  -webkit-animation-name: bounceupin; 
  -moz-animation-name: bounceupin;  
  -ms-animation-name: bounceupin;   
  -o-animation-name: bounceupin;    
  animation-name: bounceupin;      
}
```

This has to be done for all the specified CSS properties that have the -webkit prefix in the code listed in this recipe.

**CSS3 animation support in various browsers**

The minimum browser versions required to support CSS3 animations are Chrome, Firefox 5.0, IE 10, Safari 4.0 on the desktop and Android browser 4, Firefox Mobile 5.0, and Safari Mobile (iOS 2) on the mobile.

**For More Information:**

When the CSS3 property becomes a standard

The last line for each property shown in the preceding CSS is the name of the property after it becomes the standard. At this point, the browsers will drop support for that specific property's vendor prefixes. But you will not have to modify a single line of code in your CSS, as the standard property is already available in your file. The browser will skip all the properties it does not understand and pick up the standard property. So things will work just fine.

Progressive enhancement

You will notice that the transition animation in this recipe will not work properly on all the browsers. But the basic functionality of page navigation works fine everywhere. The best support for CSS3 animation, as of writing this recipe, is offered by the web kit browsers. But the beauty of CSS3 is that as browsers continue to improve and as users upgrade their devices, the user will automatically get a better experience with your app. You will not have to modify any code or make any upgrade releases. This is called Progressive Enhancement. Using jQuery Mobile means that your code is already using progressive enhancement. This would not be so easy if your app was natively written.

See also

- The Using JS to create a slide and fade page transition recipe
- The Configuring your default transitions recipe in Chapter 7, Configurations

Using JS to create a slide and fade page transition

In the previous recipe, you learned to add a custom transition to your jQuery Mobile app using CSS. You can also create custom transitions using JavaScript. This recipe shows you how to create a "slidefade" (slide and fade) effect during page transition in your app by using JavaScript.

Getting ready

Copy the full code of this recipe from the code/02/custom-js-transition sources folder. You can launch this code using the URL http://localhost:8080/02/custom-js-transition/main.html.
How to do it...

Carry out the following steps:

1. Create the customtransition.js JavaScript file and define your custom transition by adding a mycustomTransition() method, as shown in the following code snippet. Here, define how the from and to pages should animate during the transition:

   ```javascript
   function mycustomTransition( name, reverse, $to, $from ) {
     var deferred = new $.Deferred();
     // Define your custom animation here
     $to.width("0");
     $to.height("0");
     $to.show();
     $from.animate(
       { width: "0", height: "0", opacity: "0" },
       { duration: 750 },
       { easing: 'easein' }
     );
     $to.animate(
       { width: "100%", height: "100%", opacity: "1" },
       { duration: 750 },
       { easing: 'easein' }
     );
   }
   
   Next, use the standard template copied directly from the jquery.mobile.js file to complete the transition function definition:

   ```javascript
   // Standard template from jQuery Mobile JS file
   reverseClass = reverse ? " reverse" : "";
   viewportClass = "ui-mobile-viewport-transitioning viewport-" + name;
   $to.add($from).removeClass("out in reverse " + name);
   if ($from && $from[0] !== $to[0]) {
     $from.removeClass($.mobile.activePageClass);
   }
   $to.parent().removeClass(viewportClass);
   deferred.resolve(name, reverse, $to, $from);
   $to.parent().addClass(viewportClass);
   if ($from) {
     $from.addClass(name + " out" + reverseClass);
   }
   $to.addClass($.mobile.activePageClass + " " + name + " in" + reverseClass);

   return deferred.promise();
   }
   ```

2. For More Information:

3. Finally, register the custom transition with the name `slidefade` with the jQuery Mobile framework:

```javascript
// Register the custom transition
$.mobile.transitionHandlers["slidefade"] = mycustomTransition;
```

4. Next, create the `main.html` file, and include the `customtransition.js` file in the `<head>` section:

```html
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet" href="http://code.jquery.com/mobile/1.1.1/jquery.mobile-1.1.1.min.css" />
<script src="http://code.jquery.com/jquery-1.7.1.min.js"></script>
<script src="http://code.jquery.com/mobile/1.1.1/jquery.mobile-1.1.1.min.js"></script>
<script src="customtransition.js"></script>
```

5. Define the `#main` page, and include a link to open `#page2`. Use the custom `slidefade` transition with the `data-transition` attribute:

```html
<div id="main" data-role="page">
  <div data-role="header">
    <h1>Header of Main Page</h1>
  </div>
  <div data-role="content">
    <a href="#page2" data-role="button" data-transition="slidefade" data-theme="b">Go to Page 2</a>
  </div>
  <div data-role="footer">
    <h4>Footer of Main Page</h4>
  </div>
</div>
```

6. Finally, define the `#page2` page with a link to go back to the `#main` page:

```html
<div id="page2" data-role="page" data-title="Custom Transition using JS">
  <div data-role="header">
    <h1>Header of Page 2</h1>
  </div>
  <div data-role="content">
    <a href="#" data-role="button" data-rel="back" data-theme="b">Go Back</a>
  </div>
  <div data-role="footer">
  
  
  </div>
</div>
```

For More Information:

How it works...

Create the `customtransition.js` file and define the `mycustomTransition` function. Here, first create a jQuery `$`.Deferred object. Then, write your custom transition code. Set the initial width and height of the to page (or the target page) to zero. Make it visible by calling the `show()` function. Next, define the animation for the to and from pages (the from page is the current page).

The jQuery `$`.Deferred object can be used to register and invoke multiple synchronous or asynchronous callbacks, and then return their results. You can read more about this feature and the methods it provides at [http://api.jquery.com/category/deferred-object/](http://api.jquery.com/category/deferred-object/).

Call the `animate()` function and set options such as the width, height, opacity, duration of the animation, and animation curve, as shown in the code listing. Set the values, so that the from page is animated in such a way that its width and opacity become zero over the duration specified. This will slowly hide the page while sliding it to the left. Similarly, animate the to page such that in the given duration, width, height, and opacity reach 100 percent from 0. The to page fades in from the left to occupy the full screen. Now, both these animations happen together, giving a nice end result to the transition.

Once the transition is complete, the code must ensure that the right page is set as the active page. You can copy this code snippet and the other default actions that are required by the framework directly from the standard template, which is already defined in the `jquery.mobile.js` file. Now, once the transition is completed, invoke the `deferred.resolve()` function. Also return the promise of the deferred object from the transition handler.

Finally, you should register the custom transition handler with the framework using the `slidefade` name. This will ensure that when you specify the `slidefade` transition name in the `data-transition` attribute, the correct transition is picked and used from the `$.mobile.transitionHandlers` directory.

Create `main.html` and include the `customtransition.js` file in the `<head>` section. Define the #main page with a link to open #page2 using the `data-transition="slidefade"` attribute, as shown in the code. Also define #page2 with a link to go back to the #main page. You don't have to set the transition in #page2 as JavaScript already takes care of the reverse animation. Launch your app and when you navigate between the pages, you will see the new page slide in while the current page fades out, giving you the custom slide and fade transition. On second thoughts, probably "slide and shrink" would be a better name for this transition.

For More Information:
There's more...

If you have defined a custom transition in your app and used it for most of your page navigations, then you can directly set this custom transition as the default transition to be used for all your pages. This way, the `data-transition` attribute need not be specified with every link. This is specified in the `customtransition.js` file. Add the line after registering your custom transition handler (at the end of the file), as follows:

```javascript
$.mobile.defaultTransitionHandler = myCustomTransition;
```

In the preceding code snippet, `myCustomTransition` is the newly-defined transition handler. Now, all the pages would use the `slidefade` transition. But this does not affect the Dialog transitions, which use the pop transition by default.

**JavaScript transitions compared to CSS3 transitions**

Use CSS3 transitions over JS transitions in spite of the fact that you might run into vendor prefixes and incompatible browsers. With CSS3 transitions, the code required is lesser and it is much easier to develop and maintain it. Also you will not have to program the entire logic of the animation from scratch. With future versions of jQuery Mobile, the page transition framework or the logic might change, and this will break your custom JS transition.

Whereas, with CSS3, the beauty lies in the fact that your apps are progressively enhanced and fall back to basic functionality on lack of CSS3 support. As browsers improve and get upgraded, the vendor prefixes will ensure that your code works better without you modifying a single line of code. When the vendor prefixes go away, the standard attribute will get picked and everything will continue to work just fine. So use JS transitions only when you want to do more complex stuff and when CSS3 transitions would not fully support your requirement.

See also

- The Using CSS to create a bouncing page transition recipe
- The Configuring your default transitions recipe in Chapter 7, Configurations

**Using data-url to handle the login page navigation**

When you write a login page in your app, once the user enters valid credentials, you will want to redirect the user to a different page or to a different folder on success. This recipe shows you how to redirect the user to a different page during a login page navigation scenario, by using the `data-url` attribute.
Getting ready

Copy the full code of this recipe from the code/02/data-url sources folder. You can launch this code using the URL http://localhost:8080/02/data-url/login/main.html.

How to do it...

The steps to be followed are:

1. Create two folders called login and records. The login folder will contain main.html, and the records folder will contain the index.html and data.html files.

2. In the login folder, create main.html as a multi-page document. Here, first add the #main page as shown in the following code snippet. Also add a link to open the #login page.

   ```html
   <div data-role="page" id="main">
   <div data-role="header">
      <h1>Header of Main Page</h1>
   </div>
   <div data-role="content">
      <p>Page: login/main.html #main</p>
      <p><a href="#login" data-role="button">
         Login to Records folder
      </a></p>
   </div>
   </div>
   ```

3. Next, create the #login page in main.html with a link to open the index.html file. Specify the data-url attribute pointing to the records folder (for page redirection), as shown in the following code snippet:

   ```html
   <div data-role="page" id="login" data-url="http://localhost:8080/02/data-url/records/" data-title="data-url main# Login Page">
   <div data-role="header">
      <h1>Header of Login Page</h1>
   </div>
   <div data-role="content">
      <p>Page: login/main.html #login</p>
      <p><a href="index.html" data-role="button">
         Go to Index Page
      </a></p>
   </div>
   </div>
   ```

For More Information:

4. Now, create the `index.html` file in the `records` folder, as shown in the following code snippet. Add a link to open the `data.html` file here. Also set `data-url` for the page, as given in following the code:

```html
<div data-role="page"
     data-url="http://localhost:8080/02/data-url/records/"
>
    <div data-role="header">
        <h1>Header of Index Page</h1>
    </div>

    <div data-role="content">
        <p>Page: records/index.html</p>
        <p><a href="data.html" data-role="button">Go to Data Page</a></p>
    </div>
</div>
```

5. Finally, create the `data.html` file in the `records` folder. Add a link to the `index.html` file here. The `data-url` attribute is not set here, and the navigation will still work since the page redirect done earlier was successful:

```html
<div data-role="page">
    <div data-role="header">
        <h1>Header of Data Page</h1>
    </div>

    <div data-role="content">
        <p>Page: records/data.html</p>
        <p><a href="index.html" data-role="button" data-theme="b">Go to Index Page</a></p>
    </div>
</div>
```

**How it works...**

Each of the pages in the previous code listed also displays the page URL of the current page just below the page header. Keep an eye on this text, and compare it with the address shown in the browser address bar as you navigate through the pages in this recipe.

First, create the `login` and `records` folders. In the `login` folder, create the `main.html` file, which is a multi-page document. Add the #main and #login pages to it. In the #main page, add a `Login to Records` folder button to open the #login page. Next, create the #login page, and specify its `data-url` attribute as `http://localhost:8080/02/data-url/records`. Add an `Open the Index Page` button to this page, to open the index.html file located in the `records` folder. Now, when you launch the app and click on the login button, the #login page is shown. But the browser address bar will show the URL as `http://localhost:8080/02/data-url/records/`, as shown in the following screenshot. Whereas the text above the `Go to Index Page` button still says that the current page location is `login/main.html #login`.

For More Information:
This redirect occurred because the data-url attribute was used in the #login page div container. The jQuery Mobile framework updates the address bar with the value of this attribute instead of the actual URL used to fetch the page.

This is a very handy feature that allows you to perform redirects in your app. This recipe does not show the username or password being validated by the server. But in real life, the user would enter the username/password credentials in the #main page and on a successful response from the server, you can redirect the user to restricted folders and webpages. Do not redirect any unauthenticated users, and they will not be able to access any pages in the records folder.

Next, add the index.html and records.html files as given in the code. Add links to these pages to enable navigation between them. Now, in the #login page, when you click on the Open the Index Page button, the href attribute only specifies index.html in the code. But since the redirect has already occurred at this point, the index.html file from the records folder is opened. The index.html file is now the landing page here and allows you to access other pages, such as data.html, which are all located in the records folder. An alternate approach to using data-url is that you could also use the changePage() method to redirect the user to the index.html page on a successful login.

In index.html, set the data-url="http://localhost:8080/02/data-url/records" attribute to support proper navigation when the user clicks on the back or forward buttons of the browser. If this is not done, navigation will break if you click on the back button in index.html. data-url helps you set the correct value on the history stack.

You can play with the back and forward buttons of your browser to see how the address bar is updated when compared to the text shown below the header as you navigate through the app.

For More Information:
Using proper values for data-url

You can specify any value for the data-url attribute, and the same will be shown in the address bar. But you should take care to see that it is a valid reference and the browser should be able to render the page. Specifying incorrect or non-existent URLs will break the navigation when you refresh the browser or when you click on the back/forward buttons.

There's more...

jQuery Mobile sets and maintains the data-url attribute for all the pages in your app. Only the first page of your app does not require data-url, as it is always available in the DOM and can be referenced by its ID or URL. For all other pages, if the data-url attribute is not specified, it gets added with the value of the page ID by default. For external pages in the same domain, the relative path of the page is used as the value for data-url. For pages from different domains, the absolute path is used.

Using data-url as the href link

If a page div tag contains both the page ID and data-url, you can either use data-url or the page ID in the value of the href attribute value and navigate to that page.

Working with sub-hash urls

Some plugins dynamically break a page into separate pages. These pages have to be reached via deep links. These pages should have their data-url attribute specified in the following manner:

\[ \text{data-url} = \text{page.html} & \text{ui-page} = \text{subpage} \]

See also

- The Submitting a form using POST recipe in Chapter 6, Forms

Using History API to create a custom error pop up

The jQuery Mobile framework does not track dialogs in history. A dialog will thus not reappear when you click on the back button of your browser. Using a dialog for some features, for example to show an error pop up or an alert, has a minor issue that is very visible. When the dialog is opened from a page, the address bar will show the page URL suffixed with the &ui-state=dialog text. This might not be desirable to all. This recipe shows you how to use the History API and customize a regular dialog to appear, such as a pop up without any changes to the URL, making use of the History API.

For More Information:

Getting ready

Copy the full code of this recipe from the code/02/history sources folder. You can launch this code using the URL http://localhost:8080/02/history/main.html.

How to do it...

The steps to be followed are:

1. Create main.html, and add a link to open the errordialog.html file as a dialog. Also add an input button, as shown in the following code snippet:

   ```html
div id="main" data-role="page">
  div data-role="header">
    h1>Header of Main</h1>
  </div>
  div data-role="content">
    a href="errordialog.html" data-theme="b" data-role="button" data-rel="dialog">
      Dialog
    </a>
    input type="submit" value="Popup" id="linkButton" data-theme="b"/>
  </div>
  div data-role="footer">
    h4>Footer of Main</h4>
  </div>
</div>

2. Add the following script to the <head> section of main.html to open errorpopup.html as a dialog on the click event of the input button:

   ```javascript
   $("#main").live("pageinit", function(event) {
     $("#linkButton").bind( "click", function(event, ui) {
       $.mobile.changePage( "errorpopup.html", {
         changeHash: false,
         role: "dialog"
       });
     });
   });
   ```

3. Create the errordialog.html file to show a custom error message. Also add a button to go back to main.html, as shown in the following code snippet:

   ```html
div id="errordialog" data-role="page">
  div data-role="header">
    h1>Error !</h1>
  </div>
```

For More Information:  
4. Create errorpopup.html, and add the following script inside the page container. This is a regular dialog but it has a custom styled header. Remove its entry from the history stack when you click on the anchor link:

```html
<div id="errorpopup" data-role="page">
  <script>
    $('#errorpopup').live("pageinit", function(event) {
      $('a').click(function(event) {
        history.back();
      });
    });
  </script>
</div>
```

5. Then, add a custom header to the page and also add the link to go back to main.html:

```html
<div class="ui-corner-top ui-overlay-shadow ui-header ui-bar-a" role="banner">
  <h1 class="ui-title" tabindex="0" role="heading" aria-level="1">
    Error!
  </h1>
</div>

<div data-role="content">
  <p>Please correct and resubmit</p>
  <a href="main.html" data-role="button" data-theme="b">Close</a>
</div>
```

For More Information:
How it works...

Create main.html with the #main page having a link to open the errordialog.html page. Add an input submit button (id="linkButton") as shown. Next, create the errordialog.html page with a button to go back to main.html as given in the code. When you launch the app and click on the first button (Dialog), the errordialog.html page is opened as a regular dialog with the pop transition. You will see the address bar change and show the #&ui-state=dialog text at the end of the URL, as shown in the following screenshot. Close and open this dialog a couple of times, and then if you press and hold the back button, the browser’s history is displayed and you will see entries for the Error Dialog made in the history stack list:

Now, in main.html, add the given script to the pageinit event handler that gets invoked when the app starts. Here, handle the click event of the #linkButton input button, and invoke the changePage() method in the callback with the options described in the following section, to open the errorpopup.html page. Set the role option as dialog to open the page as a dialog. Also, set the changeHash option to false, to indicate that the URL hash must not be changed in the address bar when the page is opened.

Next, create errorpopup.html and add the given script inside the page container. In this script, bind the pageinit event that gets invoked when the page is initialized. Here, add an event handler for the click event of the anchor button. In this callback, invoke the history.back() method to remove the history entry made on the history stack. You should add this script in the page container, so that it gets invoked every time the page gets loaded and initialized in the DOM.

Next, add a custom header to the error pop-up page container. This custom header is the same as the one used in the Custom styling a dialog recipe, earlier in this chapter. This dialog header is customized to make it look more like a pop up and to avoid the close button, which is present by default in the dialog header. Finally, in the page content, add a button to go back to main.html.

For More Information:
Now, launch the app again and click on the second button (Popup). The custom dialog created is shown as a pop up, as shown in the following screenshot:

![Error dialog screenshot]

This pop up behaves differently from the default dialog. The **Close** icon is not present. You will note that the browser's address bar is not changed. You will also see that the **Error Popup** page title is not shown in the history list when you click and hold the browser's back button. Close the pop up and go back to `main.html`. You can click and hold the browser's back or front button to see that the pop up is never shown in the history list, whereas the dialog is listed, as shown in the following screenshot:

![History list screenshot]

**There's more...**

The History API is very easy to use, and provides additional methods with which you can handle and manipulate the history stack in the browser. You can use the `pushState()` method to add a new entry into the history. With `replaceState()`, you can replace the history entry and the URL of an existing entry in the stack. This is a very handy method and lets you manipulate the history to suit your app's needs. As shown in the code listed in this recipe, `history.back()` takes you back one step in the history, whereas `history.forward()` takes you one step forward. To go to a specific entry in the history stack, you can also use the `history.go()` method, passing it a numerical value on how many entries you want to jump. So, `history.go(-3)` will take you three entries back, and a plus value will take you three entries forward.

For More Information:

The popstate event on a dialog

Whenever you click on the back or forward buttons, a `popstate` event is thrown. This event is handled by the framework using the `onpopstate` handler, and the framework navigates to the next or previous page as desired. If `popstate` results in the target page being a dialog, the framework handles the event and does not navigate back to the dialog. Thus the dialog is not shown again when you click on the forward or back buttons in your browser.

The Popup widget

At the time of writing this recipe, jQuery Mobile v1.1.1 was used. So the error pop-up dialog created in this recipe is not a true pop up, as it still displayed in a separate page and does not hover over the original page. The `Popup` widget will be available with jQuery Mobile v1.2.0.

Then you can add a simple, true pop up using the `data-rel="popup"` attribute, as shown in the following code snippet:

```html
<a href="#myPopup" data-rel="popup">Open Popup</a>
<div data-role="popup" id="myPopup">
  <p>A simple true popup!</p>
</div>
```

You can optionally set the pop up not to be tracked in history using the `data-history="false"` attribute. You can read more about using pop ups at http://jquerymobile.com/demos/1.2.0/docs/pages/popup/index.html.

See also

- The Custom styling a dialog recipe
- The Using page initialization events recipe in Chapter 8, Events
- The Using `changePage()` to change a page recipe Chapter 9, Methods and Utilities

For More Information:
Where to buy this book


Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.