Community Experience Distilled

Mastering jQuery UI

Become an expert in creating real-world Rich Internet Applications using the varied components of jQuery UI

Vijay Joshi

[PACKT] PUBLISHING open source* community experience distilled

Mastering jQuery UI

Table of Contents

- Mastering jQuery UI
- Credits
- About the Author
- About the Reviewers
- www.PacktPub.com
  - Support files, eBooks, discount offers, and more
    - Why subscribe?
    - Free access for Packt account holders
- Preface
  - What this book covers
  - What you need for this book
  - Who this book is for
  - Conventions
  - Reader feedback
  - Customer support
    - Downloading the example code
    - Downloading the color images of this book
    - Errata
    - Piracy
    - Questions
- 1. Designing a Simple Quiz Application
  - Setting up jQuery UI
    - Downloading the required files
    - Using jQuery/jQuery UI libraries with a CDN
    - Setting up the folder structure for the JavaScript and CSS files
  - Creating the layout
    - Markup for the quiz page
    - Styling elements
  - Making the quiz functional

Displaying data on the page
Draggable country names
Droppable capital names and scoring
Accepting a draggable element
The drop event
Resetting the quiz
Improving the quiz
Summary
2. Building a Jigsaw Puzzle Game
Getting ready
Creating the layout
Creating the markup for the puzzle
Styling elements
Making the puzzle functional
Creating slices of the image
The CSS background-position property
Starting the game
Handling events for puzzle pieces
Checking for puzzle completion
Resetting the puzzle
Improving the puzzle
Summary
3. Creating a Website Tour
Getting ready
Designing the home page
Writing markup for the page
Styling elements
Making the tour functional
Initializing accordion and tooltips
Defining the dialog
Defining the tour steps

- Initializing the tour
- Displaying a tour step
- Making the Previous and Next buttons functional
- Ending the tour
- Improving the tour
- Summary
4. Creating a Tabbed News Reader
- Creating the folder structure
- Designing the page
  - Writing markup for tabs
  - Styling the content
- Getting the code structure ready
- Adding event handlers in the init method
- Displaying posts of a subreddit
  - Creating the tab structure
  - Building the DOM for posts
- Getting comments for a post
- Improving MyjqReddit
- Summary
5. Implementing CAPTCHA using Draggable and Droppable
- Creating the folder structure
- Implementing the drag and drop CAPTCHA
  - Setting a random color
  - Displaying the CAPTCHA
  - Making the drag and drop functional
  - Validating on the server
  - Improving the color CAPTCHA
- Creating the slider CAPTCHA
  - Generating minimum and maximum values for the slider
  - Making the slider functional
  - Validating the slider values

        - Improving the slider CAPTCHA
    - Creating the number CAPTCHA
        - Generating the five-digit number
        - Displaying CAPTCHA on the page
        - Adding the sortable functionality
        - Validating the number on the server
    - Summary
- 6. Creating an Event Timeline Using a Slider
    - Creating the folder structure
    - Designing the page
    - Styling the content
    - Getting the code structure ready
    - Creating the timeline markup from data
    - Implementing the timeline functionality
        - Making the slider work
            - The if block
            - The else block
        - Dragging the year window
        - Displaying event details when a year window is clicked on
        - Closing the event details window
    - Improving the timeline
    - Summary
- 7. Using jQuery UI with Google Maps API
    - Creating the folder structure
        - Getting a Google Maps API key
    - Designing the page
        - Styling the content
        - Getting the code structure ready
        - Setting minimum and maximum prices
        - Displaying hotels in accordion
        - Setting up the spinner

Displaying the map
Setting markers and infowindows in the map
Implementing the slider
Improving the functionality
Summary
8. Creating a Photo Album Manager
Creating the folder structure
Designing the page
Creating placeholders for albums and pictures
Writing markup for dialog boxes
Styling the content
Creating the JSON file for albums
Getting code structure ready
Implementing the initialize method
Filling album names
Displaying the albums
Making the pictures sortable
Initializing dialogs for edit, delete, and zoom
Handling click events for edit, delete icons, and zooming pictures
Editing, deleting, and rearranging pictures
Editing a picture name
Deleting a picture
Rearranging pictures of an album
The ajaxAlbum.php file
Improving album manager
Summary
9. Creating Widgets Using the Widget Factory
The folder structure
Creating a widget to search data in a table
Writing markup for the table
Styling the content

- Implementing the widget
  - Defining the widget structure
  - Setting up default options
  - Initializing the widget and attaching event handlers
  - Filtering the table
  - Making changes when the option value is changed
  - Destroying the widget
- Calling the widget from the page

Improving the searchable widget

Creating a widget to display a slideshow

- Writing markup for slides
- Styling the content
- Implementing the widget
  - Defining the widget structure
  - Setting up default options
  - Initializing the widget and displaying the first slide
  - Displaying slides one by one
  - Making changes when the option value is changed
  - Destroying the widget
- Calling the widget from the page

Improving the banner widget

Summary

10. Building a Color Picker with Hex RGB Conversion

Setting up the folder structure

Writing markup for the page

Styling the content

Implementing the color picker

- Defining the object structure
- The init method
- Changing colors and updating the spinner
  - The getHexColor method

        - Converting to hex
    - Improving the Colorpicker
    - Summary
- 11. Creating a Fully Functional Dashboard
    - Setting up the stage
    - Designing the page
        - Styling the content
    - Getting the code structure ready
    - Initializing the portlets
    - Implementing sharing buttons
    - Displaying Flickr photos
    - Creating a weather widget
    - Displaying posts from the reddit front page
    - Creating an image display widget
    - Improving the portlets
    - Summary
- A. Best Practices for Developing with jQuery UI
    - General Guidelines
        - Using a CDN
        - Use a customized build of jQuery UI
        - Using the jQuery UI icons
        - Be specific about elements as much as possible
        - Chain, chain, and chain
        - Cache selectors
            - Cache your loop variables
        - DOM manipulation is costly
    - Using jQuery UI Themeroller to customize a theme
- Index

Mastering jQuery UI

Mastering jQuery UI

First published: February 2015

Production reference: 1200215

Published by Packt Publishing Ltd.

Livery Place

35 Livery Street

Birmingham B3 2PB, UK.

ISBN 978-1-78328-665-2

www.packtpub.com

Credits

Author

Vijay Joshi

Reviewers

Ilija Bojchovikj

Thodoris Greasidis

Commissioning Editor

Owen Roberts

Acquisition Editor

Owen Roberts

Content Development Editor

Athira Laji

Technical Editor

Anushree Arun Tendulkar

Copy Editors

Roshni Banerjee

Merilyn Pereira

Vikrant Phadke

Project Coordinator

Harshal Ved

Proofreaders

Maria Gould

Samantha Lyon

Elinor Perry-Smith

Indexer

Mariammal Chettiyar

Production Coordinator

Manu Joseph

Cover Work

Manu Joseph

About the Author

Vijay Joshi is a web developer who lives in New Delhi, India, with his wife and daughter. Coming from a small town named Pithoragarh, Uttarakhand, in the Himalayas, he fell in love with coding while in college. He has also loved PHP and JavaScript/jQuery since his early coding days. Vijay believes that if you are passionate and enjoy your work, it becomes more of a hobby that is not boring and it never feels like a job. After freelancing for a few years, he founded a web development firm called Developers Lab along with two of his friends in New Delhi, where they build products for the travel industry and create custom web applications. Vijay is also the author of *PHP jQuery Cookbook*, *Packt Publishing*, and the technical reviewer of *PHP AJAX Cookbook* and *jQuery UI 1.8: The User Interface Library for jQuery*. When not coding, he likes to read, spend time with his family, blog occasionally at http://vijayjoshi.org, and dream about getting back in shape.

Writing a book is a long and complicated task and it requires the support and coordination of many people. I am thankful to the entire team at Packt, especially my content development editor, Athira Laji, for being so cooperative and patient with me.

A big thank you to all the technical reviewers as well, who helped me immensely in increasing the overall quality of this book.

Lastly, I am feeling both proud and excited to be able to contribute to the open source community that made me what I am today.

About the Reviewers

Ilija Bojchovikj, is a senior manager of user experience, design, and development. He has the know-how required to combine creativity and usability viewpoints resulting in world-class web and mobile applications and systems.

Ilija has more than 4 years of experience, partnering with internal and external stakeholders in discovering, building, improving, and expanding user experiences and creating and developing outstanding user interfaces.

Ilija has experience in creating cutting-edge interface designs and information architecture for websites and mobile applications through user-centered design processes by constructing screen flows, prototypes, and wireframes.

Special thanks to my girlfriend, Monika, for having the patience and letting me take yet another challenge that decreased the amount of time I could've spent with her.

Thodoris Greasidis is a senior web developer from Greece. He holds a polytechnic diploma in computer, networking, and communications engineering, and a master's in computer science.

Thodoris is part of the Angular-UI team and has made many open source contributions, with special interest in Mozilla projects.

He is a JavaScript enthusiast and loves bitwise operations. His interests also include web applications and services and artificial intelligence, especially multiagent systems.

Special thanks to my family and fiancée, who supported me while reviewing this book, and to the guys from work who encouraged me to get involved.

www.PacktPub.com

To Tanu and Nauni, with all my love.

Preface

jQuery UI needs no introduction for itself. As the official website says, “jQuery UI is a curated set of user interface interactions, effects, widgets, and themes built on top of the jQuery JavaScript Library.” It is very likely that any developer who has worked with jQuery would have used at least one of the jQuery UI widgets.

Since it is a large library having many components with each component having multiple options, it is common for developers to get confused. Despite having excellent documentation, there are only a few resources that describe jQuery UI using its practical applications. To become an expert, you must know how jQuery UI fits with real world applications and which of its components have to be used when a use case is provided.

This is the goal of this book. Our aim is to improve your knowledge of jQuery UI to a master level, so that you can use it in complex real world projects with ease.

Each chapter of the book is a mini project in itself. There are detailed step-by-step instructions along with helpful pictures that will guide you through each chapter. The chapters are not arranged in any particular order, so you may pick up any one at random.

I am optimistic that this book will help you take jQuery UI skills to the next level.

Happy Coding!

What this book covers

Chapter 1, *Designing a Simple Quiz Application* , makes use of jQuery UI's interaction components to create a quiz application. You will learn about jQuery UI's sortable, draggable, and droppable components in this chapter.

Chapter 2, *Building a Jigsaw Puzzle Game*, teaches you to create a Jigsaw puzzle game using some of the jQuery UI's interaction components. You will learn to divide an image into multiple tiles along with sortable techniques.

Chapter 3, *Creating a Website Tour*, will create a user-friendly tool to tour different parts of a web page. You will learn about jQuery UI's tooltip component and some other components in this chapter.

Chapter 4, *Creating a Tabbed News Reader*, teaches you to create a news reader using an external API. Using the reddit API, you will learn about creating a mashup with jQuery UI's tabs and dialog components.

Chapter 5, *Implementing CAPTCHA using Draggable and Droppable*, deals with creating CAPTCHAs. Using jQuery UI's draggable and droppable components, you will be able to create three different CAPTCHA implementations.

Chapter 6, *Creating an Event Timeline Using a Slider*, will teach you to create an interactive timeline to visualize events that occurred in different years. You will also learn different techniques of using the slider component.

Chapter 7, *Using jQuery UI with Google Maps API*, teaches you to use jQuery UI components with Google Maps. You will learn to control markers on a map using a slider. You will also learn to control the zoom level using a spinner.

Chapter 8, *Creating a Photo Album Manager*, explains how to create a photo album to display pictures. Users will be able to edit the names of any picture as well as delete and/or rearrange pictures in an album.

Chapter 9, *Creating Widgets Using the Widget Factory*, teaches you to create new widgets. In detailed steps, you will learn to create two different widgets to cover all aspects of the jQuery UI widget factory.

Chapter 10, *Building a Color Picker with Hex RGB Conversion*, creates a simple color selector tool. Along with RGB to Hex conversion, this chapter will guide you in creating a tool that will allow you to change the text as well as the background color of a page, using multiple slider and spinner components.

Chapter 11, *Creating a Fully Functional Dashboard*, puts together all the learning acquired through the previous 10 chapters to create a dashboard with multiple widgets. You will learn to create six different widgets in this chapter.

Appendix, *Best Practices for Developing with jQuery UI*, guides you with the best possible ways to create rich applications. This chapter will also teach you about the best coding practices and optimizations.

What you need for this book

You should have a web server installed on your system. Apache is recommended but you can use IIS as well. You will also need PHP (version 5.2 or above) for some chapters. You can install all of these in a single go using a software such as Wamp Server or you can install them separately. jQuery (version 1.6 or higher recommended) and jQuery UI libraries (version 1.11.2 recommended) will also be required.

In terms of technical proficiency, this book assumes that you have worked with jQuery and jQuery UI libraries, HTML, CSS, and JSON. This book will take care of the rest.

Who this book is for

This book is aimed at frontend developers who use jQuery and jQuery UI on a daily basis and want to take their skills of jQuery UI to an advanced level. The book provides step-by-step instructions, with details supported by pictures to help you become an expert in using different jQuery UI components.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "We have created a div with CSS class `container` which will act as parent div for all the page elements"

A block of code is set as follows:

```
var t = this;
  $( ".slider" ).slider(
  {
    range: "min",
    max: 255,
    slide : function (event, ui)
    {
      t.setColor($(this), ui.value);
    },
    change : function (event, ui)
    {
      t.setColor($(this), ui.value);
    }
  });
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
var t = this;
  $(".slider").slider(
  {
    range: "min",
    max: 255,
    slide : function (event, ui)
    {
      t.setColor($(this), ui.value);
    },
    change : function (event, ui)
    {
      t.setColor($(this), ui.value);
    }
  });
```

Any command-line input or output is written as follows:

```
# cp /usr/src/asterisk-addons/configs/cdr_mysql.conf.sample
     /etc/asterisk/cdr_mysql.conf
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "clicking the **Next** button moves you to the next screen".

Note

Warnings or important notes appear in a box like this.

Tip

Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to <`feedback@packtpub.com`>, and mention the book title via the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on www.packtpub.com/authors.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

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.

Downloading the color images of this book

We also provide you a PDF file that has color images of the screenshots/diagrams used in this book. The color images will help you better understand the changes in the output. You can download this file from: http://www.packtpub.com/sites/default/files/downloads/2018_6652OS_ColorImages.pdf.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting http://www.packtpub.com/submit-errata, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from https://www.packtpub.com/support.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at <`copyright@packtpub.com`> with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at <questions@packtpub.com> if you are having a problem with any aspect of the book, and we will do our best to address it.

Chapter 1. Designing a Simple Quiz Application

To begin with, we will design a simple quiz application where users have to match country names to their corresponding capital names by dragging a country name over its correct capital. A correct match will result in one point, and drag and drop will be disabled for both the country and its capital.

The display will contain two columns, the first one will have country names and the second one will have the names of the capitals of those countries. Both the columns will be shuffled randomly. Once the user has matched all the names successfully, a modal dialog will appear. Users will also be given the option to reset the quiz. Resetting will restart the game and shuffle the lists again.

Setting up jQuery UI

We will need the jQuery and jQuery UI libraries before going ahead. Throughout this book, we'll use jQuery Version 1.10.2 and jQuery UI Version 1.10.4 with the `UI lightness` theme . Note that the jQuery UI files also contain a copy of the jQuery source file.

Downloading the required files

To download jQuery, visit the download page on the jQuery website at http://jquery.com/download/.

Tip

Make sure that you download the correct version as per your requirements.

For jQuery UI, visit the download page at http://jqueryui.com/download/ to download the jQuery UI library.

Tip

Since we will cover all components throughout the book, download the full version of the library.

Using jQuery/jQuery UI libraries with a CDN

You might be aware that **Content Delivery Networks** (**CDN**) host many popular libraries. Since the browsers cache JavaScript files, your page will not have to load a JavaScript file again if it is referenced from a CDN and already cached in browser. You can link jQuery and jQuery UI among CDN's many other libraries.

Tip

Make sure that you are connected to the Internet if you have referenced the libraries from the CDN in your pages.

Google, Microsoft, and some other companies provide CDN for jQuery, jQuery UI, and other libraries. Here are the links to pages for downloading these libraries:

- Google CDN can be downloaded from https://developers.google.com/speed/libraries/devguide#jquery.
- Microsoft CDN can be downloaded from http://www.asp.net/ajaxlibrary/cdn.ashx#Using_jQuery_from_the_CDN_21.
- CDNJS can be downloaded from https://cdnjs.com/. It is a helpful site where you can find many libraries and their different versions.

Setting up the folder structure for the JavaScript and CSS files

We will now set up the folder structure that we will use for all the chapters in this book. The steps are as follows:

1. In your document root, create a folder named `MasteringjQueryUI`. Then, create a folder for each chapter inside it.
2. For this chapter, create a new folder named `Chapter1` inside `MasteringjQueryUI` and two more folders named `js` and `css` inside the `Chapter1` folder.
3. Now extract the jQuery UI files into a separate folder and go to its `js` folder. You will see three files: jQuery source file and full and minified versions of jQuery UI.
4. Copy the jQuery source file and any one version of jQuery UI source files and place them inside the `js` folder that you created inside `Chapter1` of the `MasteringjQueryUI` folder.
5. Also, copy the `ui-lightness` folder from the `css` folder from the downloaded jQuery UI to the `css` folder of `Chapter1`.

Now we are ready to experiment with jQuery UI and create some informative and fun examples. Let’s start our journey by creating the quiz application.

Creating the layout

In the newly created folder `Chapter1`, create a file named `index.html` and another `.js` file named `quiz.js` inside the `js` folder of `Chapter1`. The `quiz.js` file will contain all the code that we need to make the quiz functional.

Markup for the quiz page

Open the `index.html` file for editing using your favorite text editor, and write the following code in it:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Designing a simple quiz application</title>
    <link rel="stylesheet" href="css/ui-lightness/jquery-ui-
1.10.4.custom.min.css">
  </head>
  <body>
    <div class="container">
      <span id="score"></span>
      <button id="reset" type="button">Reset</button>
      <div class="clear"></div>
      <hr/>
      <div id="leftCol">
        <ul id="source">
        </ul>
      </div>
      <div id="rightCol">
        <ul id="target">
        </ul>
      </div>
    </div>

    <div id="dialog-complete" title="Well Done!">
      <p><span class="ui-icon ui-icon-check"></span>
      Well done. You have completed the quiz successfully.</p>
    </div>

    <script src="js/jquery-1.10.2.js"></script>
    <script src="js/jquery-ui-1.10.4.custom.min.js"></script>
    <script src="js/quiz.js"></script>


  </body>
</html>
```

Tip

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.

In the preceding code, inside the `head` section, we referenced the jQuery UI `.css` file. If you have placed the `.css` file elsewhere, make sure to correct the path accordingly. The path can either be an absolute path or a relative path.

Inside the body, we have a `div` element named `container` that wraps our entire markup. Inside `container`, we have created two `span` elements. The first `span` has the `id` value `score` and it will be used to show the score of the user. The second `span` has the `id` value `reset` and it will be used to reset the quiz to its initial state.

After this, we have to create two more `div` elements having the `id` value `leftCol` and `rightCol`, respectively. `leftCol` has an `ul` element with the `id` value `source` inside it. This `ul` element will contain the names of countries as list items. Similarly, `rightCol` has another `ul` element with the `id` value `target` inside it. It will have the names of capitals as list items.

After the `container` element, we have created yet another `div` element with the `id` value `dialog-complete`, which will be displayed once the user has completed the quiz successfully. Inside the `dialog-complete` element, we have placed a success message.

Finally, at the bottom of the page, reference the jQuery, jQuery UI, and `quiz.js` files.

Note

Make sure that jQuery is included first before jQuery UI, and that any other JavaScript file or custom jQuery code is included or written.

Styling elements

We will also need some CSS styling to make the elements look good. In the `head` section of the `index.html` file, write the following code:

```
<style type="text/css">
  body{
    font-family:arial,verdana;
    font-size:12px;
    margin: 0px auto;
    width: 600px;
  }

  div.container{
    border: 1px solid #000;
    float:left;
    margin:0 auto;
    padding:10px;
    width: 100%;
  }

  #leftCol{
    float:left;
  }

  #rightCol{
    float:right;
  }

  ul{
    list-style:none;
    margin:0;
    padding:0;
    width:50%;
  }

  li{
    border:1px solid #000;
    font-weight:bold;
    margin:5px 0;
    padding:10px 0;
    text-align:center;
    width:175px;
  }

  #source li{
    cursor:move;
  }

  #score{
    font-weight:bold;
    float:left;
    color:#ff0000;
  }
  #reset{
    color:#ff0000;
    cursor:pointer;
    font-weight:bold;
    text-align:right;
    text-decoration:underline;
    float:right;
  }

  .clear{
  clear:both;
  }

  #dialog-complete{
    display:none;
  }

  #dialog-complete span{
    float:left;
    margin:0 7px 20px 0;
  }
</style>
```

In the preceding code, first we defined some basic styles for the `body` and the `container` elements. After that, the styles were defined for the `ul` element and its `li` items. These styles will display the list items in the form of individual boxes. CSS for the `score` and `reset` items follow next and finally some basic styling for the dialog elements.

Making the quiz functional

Our UI part is now complete and we can proceed to make the quiz functional. We will do this in a few steps. First of all, we will display the data on the page in two columns. Then, we will make the country names draggable. Finally, the list items with the capital names will be made droppable so that we can drop a country name inside a capital. We will also have to ensure that a droppable capital name accepts only the correct country name. Finally, the resetting logic will be built.

Displaying data on the page

Open the `quiz.js` file for editing and write the following code:

```
$(document).ready(function()
{
  createQuizLayout();
});
```

On the document ready event we call a function named `createQuizLayout` which we need to define now.

```
function createQuizLayout()
{
  //declare arrays of countries and their capitals.
  var countries = ["USA", "UK", "India", "Germany", "Turkey", "France",
"Nepal", "Japan", "South Africa", "Maldives"];
  var capitals = ["Washington", "London", "Delhi", "Berlin", "Istanbul",
"Paris", "Kathmandu", "Tokyo", "Capetown", "Male"];

  var arrCountry = [];
  for(var i=0; i<countries.length; i++)
  {
    arrCountry.push('<li data-index="' + (i+1) + '">' + countries[i]
+'</li>');
  }

  var arrCapital = [];
  for(var i=0; i<capitals.length; i++)
  {
    arrCapital.push('<li data-index="' + (i+1) + '">' + capitals[i]
+'</li>');
  }

  //shuffle the arrays
  arrCountry = shuffle(arrCountry);
  arrCapital = shuffle(arrCapital);
  // once country and capital items are ready, we insert them into DOM
  $('#source').html(arrCountry.join(''));
  $('#target').html(arrCapital.join(''));
}
```

Here is what the preceding code does:

- We have defined two arrays named `countries` and `capitals`.
- The `countries` array contains names of 10 countries and the `capitals` array contains names of the capitals of the countries defined in the countries array. The names of capitals must be in the same order as their respective countries.
- Since we want to display the names of countries and capitals in a random order, we will create two arrays and fill them with list items and shuffle them.
- We started with country first. We declared an array named `arrCountry`. Then, we loop in the `countries` array and create a list item with the country name and push it into the `arrCountry` array.

- The same process is repeated for the `capitals` array.

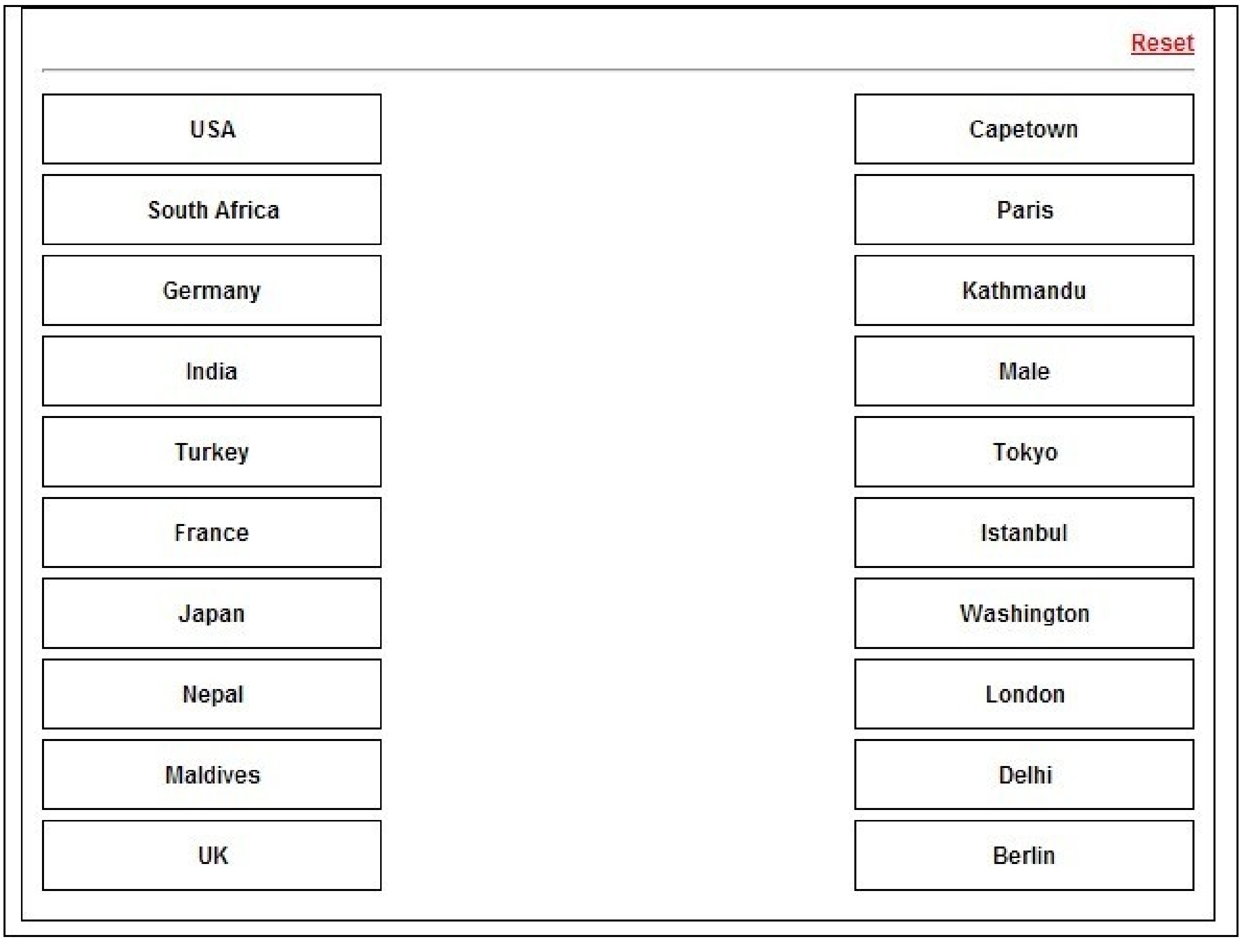
An important point to note here is that we are giving a data attribute named `index` to each list item having a value from 1 to 10. Since we have both the countries and capital names in the same order, `index` will be used to match which country belongs to which capital.

After both arrays are populated, we will shuffle them so that the order of countries and capitals becomes random. For this, we will use a simple `shuffle` function from the website http://jsfromhell.com/array/shuffle. The `shuffle` function is defined as follows:

```
function shuffle(o)
{
  for(var j, x, i = o.length; i; j = Math.floor(Math.random() * i), x = o[-
-i], o[i] = o[j], o[j] = x);
    return o;
};
```

After calling the `shuffle` function on both arrays `arrCountry` and `arrCapital`, the array elements are inserted in **DOM** after combining them into a single string using the JavaScript `join` function. The elements in the array `arrCountry` are inserted in `ul` with the `id` value `source` and those in the array `arrCapital` are inserted in `ul` with the id value `target`.

Open your browser and point it to the `index.html` file of the `Chapter1` folder now. You will see a page similar to the one shown in the following screenshot:



If you reload the page, you will see that the order of countries and capitals changes each time. This is because shuffling creates a new order for items of both lists.

Draggable country names

To make the country names draggable, we will use the `draggable` component of jQuery UI. As the name suggests, the `draggable` component allows DOM components to be moved around using a mouse. To do this, go to the `$(document).ready()` section of our `quiz.js` file and call another function named `initQuiz`. The `$(document).ready()` callback function should look like this now:

```
$(document).ready(function()
{
  createQuizLayout();
  initQuiz();
});
```

Now define the `initQuiz` function outside document ready handler as follows:

```
function initQuiz()
{
  $('#source li').draggable(
  {
    revert : true,
    revertDuration: 200,
    cursor: "move"
  });
}
```

The preceding code calls the `draggable` method of the jQuery UI library. It is being called upon the `li` elements of the `ul` source, which means it will make all the list items draggable inside the source `ul`. Further, we are also giving the `draggable` method three options that we need for our application: `revert`, `revertDuration`, and `cursor`. Let's look at these in more detail:

- `revert`: This decides whether the element being dragged should revert to its original position or not. In our case, we will set it to `true`. We will drag a country name onto a capital name and revert it to its original position, that is, the country list. Another possible value for `revert` is `false`, which means it will stay at the place where it is when dragging stops. The values, `valid` and `invalid`, can also be provided (as strings) for the `revert` option. The value `valid` means the draggable object will revert only if the draggable object has been dropped on a droppable element. The value `invalid` means the `draggable` fuction will revert if the draggable object has not been dropped. Alternatively, a function can also be provided to revert. This is required in complex cases where we need to perform any manipulations. The `return` value for this function will decide if it will revert or not. If `true` is returned, the element will revert.
- `revertDuration`: This defines the duration for the `revert` option in milliseconds. The lower the value, the faster it will revert. This value is not considered if the revert option is set to `false`.
- `cursor`: This is the style of cursor while an element is being dragged.

Our draggable elements are ready now, and so it is time to make the capital names droppable and build the logic to match countries to their correct capitals.

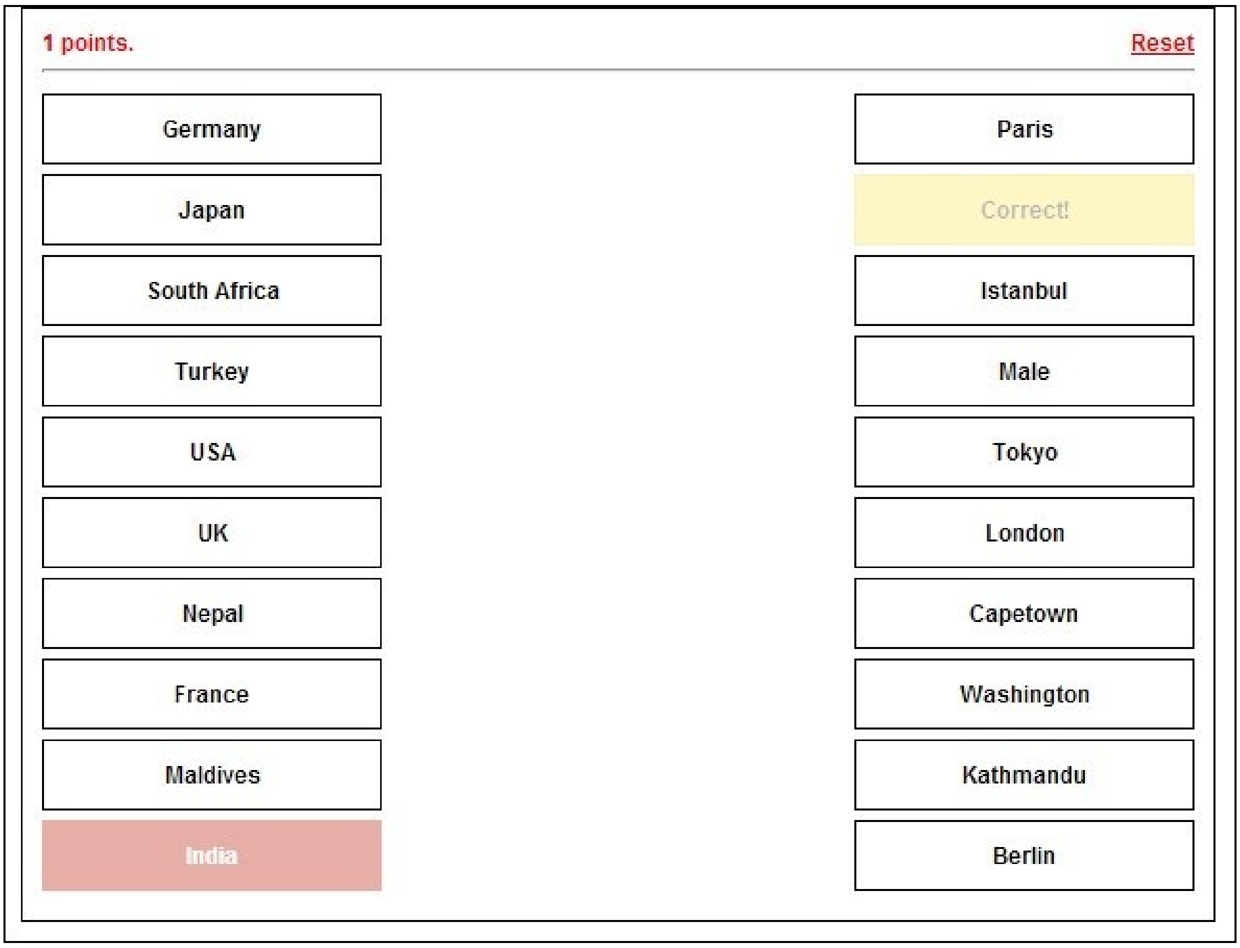
Droppable capital names and scoring

In the previous section, we created an `initQuiz` function where we made our countries draggable. After the draggable code, write the following code to make the capitals droppable:

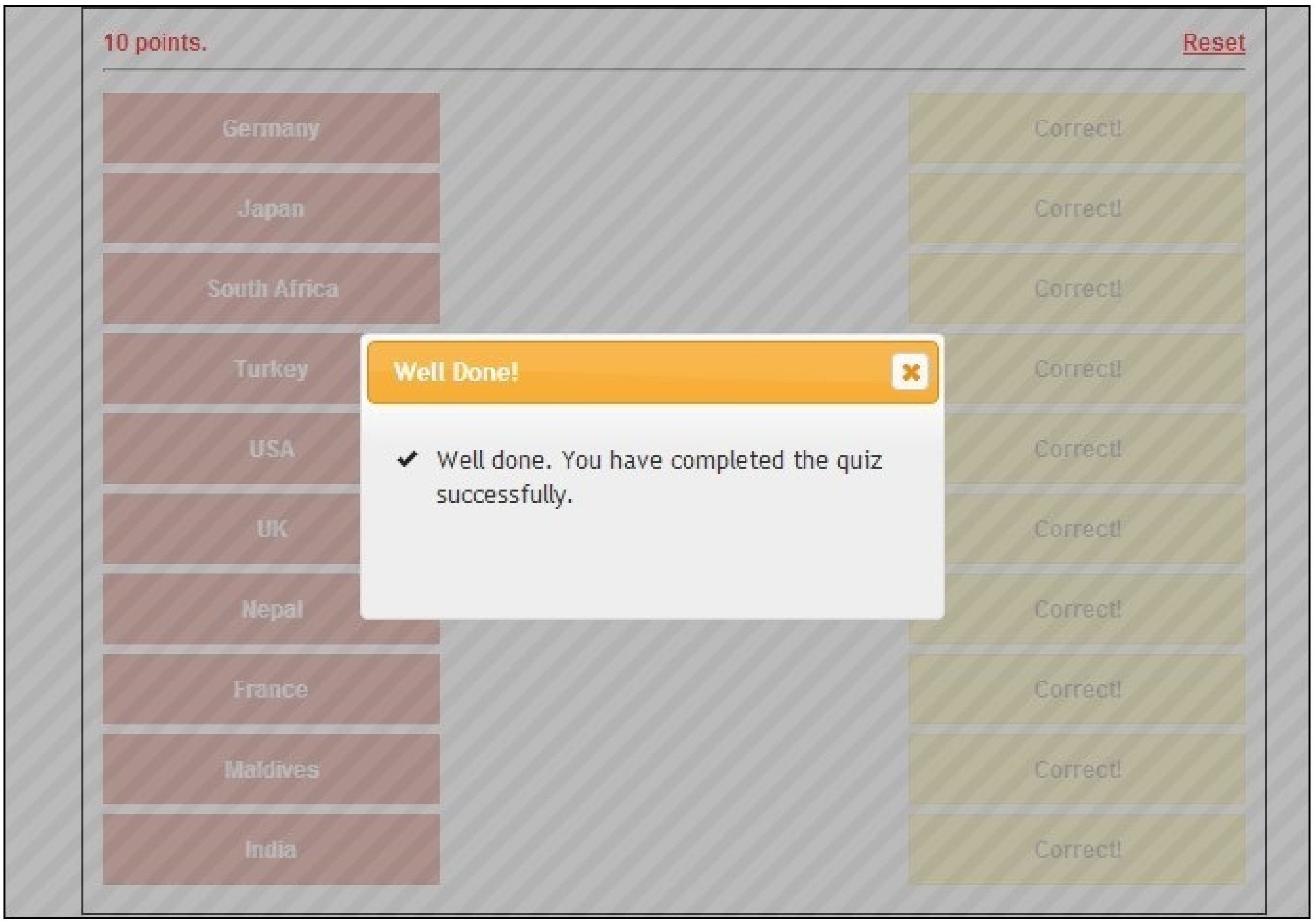
```
var totalScore = 0;
$('#score').text(totalScore + ' points.');
$('#target li').droppable(
{
  accept : function(draggable)
  {
    if(parseInt(draggable.data('index'), 10) ===
parseInt($(this).data('index'), 10))
    {
      return true;
    }
    else
    {
      return false;
    }
  },
  drop: function( event, ui )
  {
    var that = $(this);
    that.addClass( "ui-state-highlight" ).html( 'Correct!'
).effect('bounce');
    that.droppable('disable');
    ui.draggable.addClass('correct ui-state-error');
    (ui.draggable).draggable('disable');

    totalScore++;
    $('#score').text(totalScore + ' points.');
    if($('li.correct').length == 10)
    {
      $( "#dialog-complete" ).dialog({
        resizable: false,
        modal: true
      });
    }

  }
});
```

Now save the `quiz.js` file and refresh your browser. You will be able to drag the country names now. Drag a country name to its correct capital and you will see that the country will revert to its original position. The capital list item will show a bounce effect and its text will change to **Correct!**. Both the country and capital names will be disabled now. You will not be able to drag the country name as well. On the top left hand side, the page will show the score as **1 points**.

The screen will look like the following screenshot:



Try the drag and drop for all countries in the left-hand side list. When you have matched all countries correctly, you will see a dialog box and the page will look like the following screenshot:



Well done. You have completed the quiz successfully.

So, a lot is happening in the preceding code. We will look at it step by step.

We defined a variable named `totalScore` and set it to `0`. We also inserted the score inside the HTML element with the `id` value `score`. Each time the quiz starts, the score will be reset as well. After this, we call the `droppable` method of jQuery UI on the list items of `ul` with the `id` value `target` to make them ready to accept the draggable country elements.

We are using the `accept` option of the jQuery UI `draggable` method to check for the correct matches of country and capital, and we are using the `drop` event to change the UI and scoring.

Accepting a draggable element

The `accept` option of a `droppable` method defines which draggable element will be accepted by the `droppable` method when a draggable element is over it; either a jQuery selector or a function can be provided for this purpose. If a selector is given, only the draggable element matching that selector will be accepted by the `droppable` method. Since we want to match an individual country to its capital, it is better for us to use a function instead. The function will receive the current draggable element that is being dragged as a parameter. Inside the function, `$(this)` will refer to the current droppable element. The code is as follows:

```
if(parseInt(draggable.data('index'), 10) == parseInt($(this).data('index'),
10))
  {
    return true;
  }
  return false;
```

Since we have already defined data attributes for both countries and capitals, we can match those to check if the current draggable-droppable pair is a correct country-capital pair or not. If the indexes match, we return `true`; otherwise, we return `false`.

A return value `true` means the `droppable` method will accept the draggable element, and will allow the draggable element to be dropped in it.

The drop event

The `drop` event will receive a draggable element once it has been passed from the `accept` option. If the `accept` option returns `false` for any draggable element, then the `drop` event will not be called. In our case, this means we will only receive a country's draggable element and its corresponding capital's droppable element.

The callback function for the `drop` event receives two parameters: `event` and `ui`. Of these two, we are interested in the `ui` object. Among other values, it provides us with a reference to the draggable element that was dropped. To refer to the current droppable element where the draggable element is dropped, we have `$(this)` variable with us. The code is as follows:

```
$( this ).addClass( "ui-state-highlight" ).html( 'Correct!'
).effect('bounce');
$( this ).droppable('disable');
```

In the preceding code, we added the jQuery UI framework's CSS class `ui-state-highlight` to the current droppable element and then changed that list item's HTML content to **Correct!** and added the `bounce` effect to the droppable capital.

Since the droppable capital has been matched successfully with its country, we no longer need it as a droppable element. Hence, the preceding code uses the `disable` method of the droppable component to disable the droppable functionality.

The next two lines add CSS classes named `correct` and `ui-state-error` to the `draggable` method and then disable it. The code is as follows:

```
ui.draggable.addClass('correct ui-state-error');
(ui.draggable).draggable('disable');
```

The `correct` class will be used to determine how many successful countries have been matched. The class `ui-state-error` is just for presentation purposes to make the successfully matched country name highlighted. Using the draggable `disable` method, we disable the specific draggable element as well, because it has been matched and we do not want it to be dragged again.

Since the `drop` event receives only the accepted draggable elements, we can safely increase the variable `totalScore` by `1` and insert the new value back to the DOM in the element `score`. This shows us the latest score each time a new match is made.

Finally, we count the number of list items in the countries' column that have the CSS class named `correct` associated with them. Since we have 10 elements, if all the 10 list items have the CSS class `correct` attached to them, it means the quiz is complete. We then show a jQuery UI dialog component that we kept hidden in our HTML page initially.

Resetting the quiz

If you were wondering why we created the functions `createQuizLayout` and `initQuiz` when we wrote the code without them, the answer is that we need to call them again. It is better not to repeat yourself. We can now reset the quiz without having to reload the page.

We have already created an element with id `reset`. Visit the `$(document).ready()` callback again and write the following code after those two function calls. The section will now look like this:

```
$(document).ready(function()
{
  createQuizLayout();
  initQuiz();

  $('#reset').on('click', function()
  {
    $('#source li').draggable('destroy');
    $('#target li').droppable('destroy');
    createQuizLayout();
    initQuiz();
  });
});
```

We have an event handler registered at the click of the `reset` button. It is using the `destroy` method of jQuery UI on the draggable and droppable elements. The `destroy` method will remove the complete draggable and droppable functionality from respective elements. It will also remove any special CSS classes that jQuery UI might have applied earlier.

After bringing the page to its initial state, we call the `createQuizLayout` and `initQuiz` functions again, which will initialize our quiz once more.

Improving the quiz

This was a basic application to begin with. There can be many more enhancements to this quiz to make it more feature rich. You are encouraged to add more features to it and refactor the code as well.

Here are some ideas that you can start with:

- Remove successfully matched countries and capitals
- If you watch closely, we do not need the variable `totalScore`

  ### Tip

  List items with the class `correct` are enough for calculating scores.

- Allow negative scoring if the user drops the country in an incorrect capital

Summary

The `draggable` and `droppable` methods are important components of jQuery UI in order to make interactive applications. We explored a few options presented by these two components and created a simple quiz application in this process. We will see more options presented by these two components in the following chapters as well, where we will create a jigsaw puzzle game.

Chapter 2. Building a Jigsaw Puzzle Game

Moving forward a bit more, in this chapter, we will see more (and slightly complex) use cases of draggable and droppable components. We will develop a jigsaw puzzle game where we will divide a picture into small pieces of the same size, and the user will have to rearrange these small pieces by dragging and dropping them to form a complete picture.

The UI will have two containers on the page. One container will be used to keep the puzzle pieces, and the other will act as a canvas for arranging these pieces. Initially, users will be displayed a picture along with a **Start** button. Once the **Start** button is clicked, the image will be divided into 16 pieces and these pieces will be placed in a separate container. Users will have to drag and drop the individual pieces on the canvas and arrange them to make the complete picture.

Getting ready

Make sure that you have set up jQuery and jQuery UI files as explained in Chapter 1, *Designing a Simple Quiz Application*. You will also need an image that has dimensions equal to 400px x 400px. The code in this chapter uses an image of a cute cat, named `kitty.jpg`. Of course, you can use any image of your choice. Keep this image in the `Chapter2` folder.

Since we will create 16 pieces of this image, each having both width and height equal to 100 px, keep in mind that the image you choose should not have a single square area of the same color exceeding 100px x 100px. This will create problems while solving the puzzle as you will be unable to determine the correct locations of multiple pieces of the same color.

Creating the layout

Create a file named `index.html` inside the `Chapter2` folder. It will contain the entire HTML. For the JavaScript code, create another file named `puzzle.js` inside the `js` folder of `Chapter2`. Now the `Chapter2` folder will have four items: the `index.html` file, the image file for the puzzle (`kitty.jpg`), the `js` folder, and the `css` folder.

Creating the markup for the puzzle

Our first step towards creating the puzzle will be to prepare the required HTML markup. This HTML markup will then be styled using CSS. The following HTML markup will prepare the bare-bones structure of the page required to make the puzzle:

```
<html>
  <head>
    <meta charset="utf-8">
    <title>Designing a Jigsaw Puzzle</title>
    <link rel="stylesheet" href="css/ui-lightness/jquery-ui-
1.10.4.custom.min.css">
  </head>
  <body>
    <div class="container">
      <div id="pieceBox"></div>

      <div id="puzzleContainer"></div>

      <class="clear">&nbsp;</div>
      <div class="clear"> </div>
      <span id="message"></span>

      <div class="clear"> </div>

      <ul class="buttons">
        <li><button id="start">Start</button></li>
        <li><button id="reset">Reset</button></li>
      </ul>
    </div>

    <script src="js/jquery-1.10.2.js"></script>
    <script src="js/jquery-ui-1.10.4.custom.min.js"></script>
    <script src="js/puzzle.js" type="text/javascript"></script>
  </body>
</html>
```

In the preceding markup, we have a `div` element with the value of `id` as `container`, which is a wrapper for whole page. Inside it are two `div` elements with `pieceBox` and `puzzleContainer` as the values for `id`. The element `pieceBox` will act as a box in which we will keep the 16 sliced pieces of the image, whereas the element `puzzleContainer` will be the canvas on which users will drop these pieces and arrange them. Next, there is a `span` element where we will show a success or error message after the user has placed all the pieces. There is also a `list` containing two list items. We will use them as **Start** and **Reset** buttons.

Finally, at the bottom of the page are the references to the jQuery, jQueryUI, and `puzzle.js` files.

Tip

Ensure that the path for the CSS and JavaScript files is correct.

Styling elements

After our page structure is ready, we need to add CSS styles for different elements created in the markup to give the page the desired appearance. This will be done by adding some CSS rules to the `head` section of the `index.html` file. The styles that we will use are explained here:

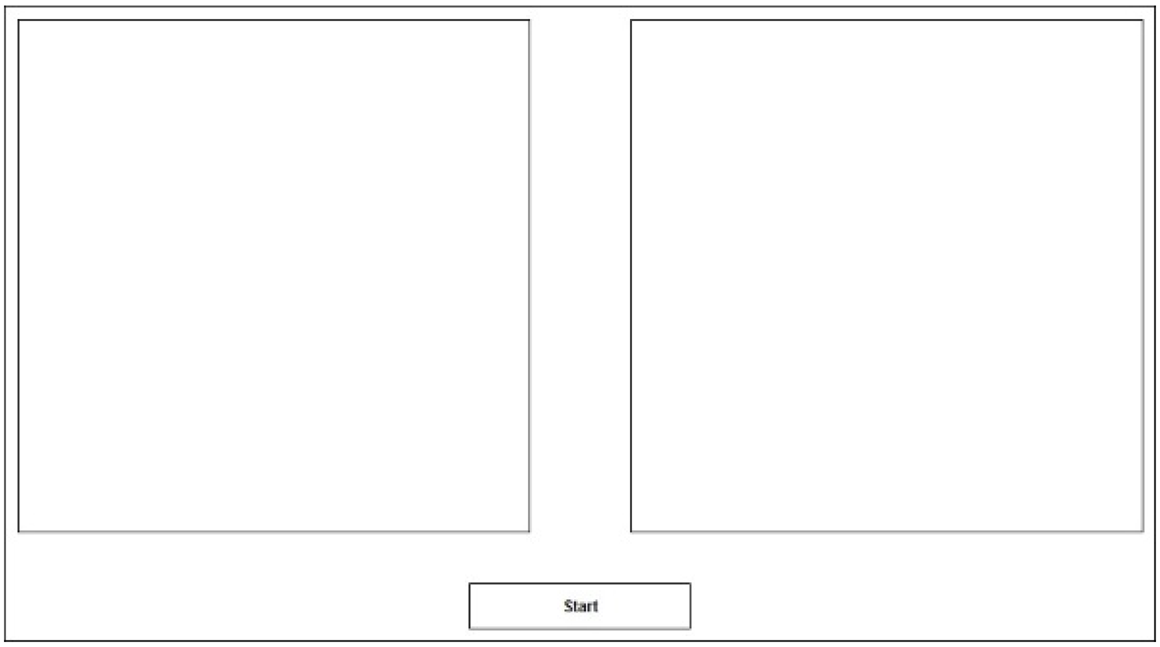
```
<style type="text/css">
  body{
    font-family:arial,verdana;
    font-size:12px;
    margin: 0 auto;
    width: 900px;
  }
  div.container{
    border: 1px solid #000;
    float:left;
    margin:0 auto;
    padding:10px;
    width: 100%;
  }

  #pieceBox{
    border: 1px solid #000;
    float: left;
    height: 408px;
    margin: 0 auto;
    position:relative;
    width: 408px;
  }

  #puzzleContainer
  {
    border: 1px solid #000;
    float: right;
    margin: 0 auto;
    height: 408px;
    width: 408px;
  }

  div.img{
    background-image: url('kitty.jpg');
    background-repeat: no-repeat;
    height:100px;
    width:100px;
    float:left;
    border:1px solid #000;
  }
  ul{
    text-align:center;
    list-style:none;
    margin:0;
    padding:0;
  }
  span#message{
    clear:both;
    display: none;
    font-size: 20px;
    padding: 20px 0;
    text-align: center;
    width: 100%;
  }

  ul.buttons{
    cursor:pointer;
    margin-top:10px;
  }

  ul button{
    border:1px solid #000;
    font-weight:bold;
    margin:0 auto;
    padding:10px 0;
    text-align:center;
    width:175px;
    display:inline-block;
  }
  #reset{
    display:none;
  }
  .clear{
    clear:both;
  }
</style>
```

First, we defined some basic styles for `body`, `container`, `pieceBox`, and `puzzleContainer`. Then we defined styling for the `div` elements that have the `.img` class. This class will be applied to the pieces of the puzzle. Since it will not be efficient to create 16 different images to use as jigsaw pieces, we will use a single image as a sprite. Therefore, we set the `background-image` property to `kitty.jpg`, which is the image that we are going to use. Using the background-position CSS property, we will be able to show a specific 100 px x 100 px part of the image.

After this, we defined some CSS properties for the success or error message and the buttons. In the last CSS rule, we hid the **Reset** button as it will not be required initially.

After writing the HTML and markup, we are ready to make the puzzle functional by plugging in the JavaScript to create the game.

Meanwhile, run the `index.html` file in your browser and you will see a screen with two boxes, as shown in the following screenshot. Based on their IDs, we will call these boxes `pieceBox` and `puzzleContainer`, respectively:


Start

Making the puzzle functional

Before writing any JavaScript code to create a functional puzzle, let’s write down the features of our puzzle and see how we will achieve them.

When the page loads, an image will be displayed to the user in `puzzleContainer`, and a **Start** button will be displayed under it. The image will actually be a collection of 16 different div elements, each having the same background image but a different background position. Using the `background-position` CSS property, we will be able to display the complete image to the user. Once the **Start** button is clicked, we will take these 16 images and place them at random positions inside pieceBox. We will also display a 4 x 4 grid, `puzzleContainer`, where any of the 16 pieces could be dropped. We will then attach appropriate event handlers that will allow us to drag an individual puzzle piece from `pieceBox` to `puzzleContainer`. Once a piece has been moved to `puzzleContainer`, it cannot be dropped back to `pieceBox`. It can, however, be dragged into any other cell in `puzzleContainer`. Once the user has arranged all the pieces, a relevant message will be displayed.

Enough with the theory for now! Let’s dive into some practical JavaScript. In your text editor, open the `puzzle.js` file.

Creating slices of the image

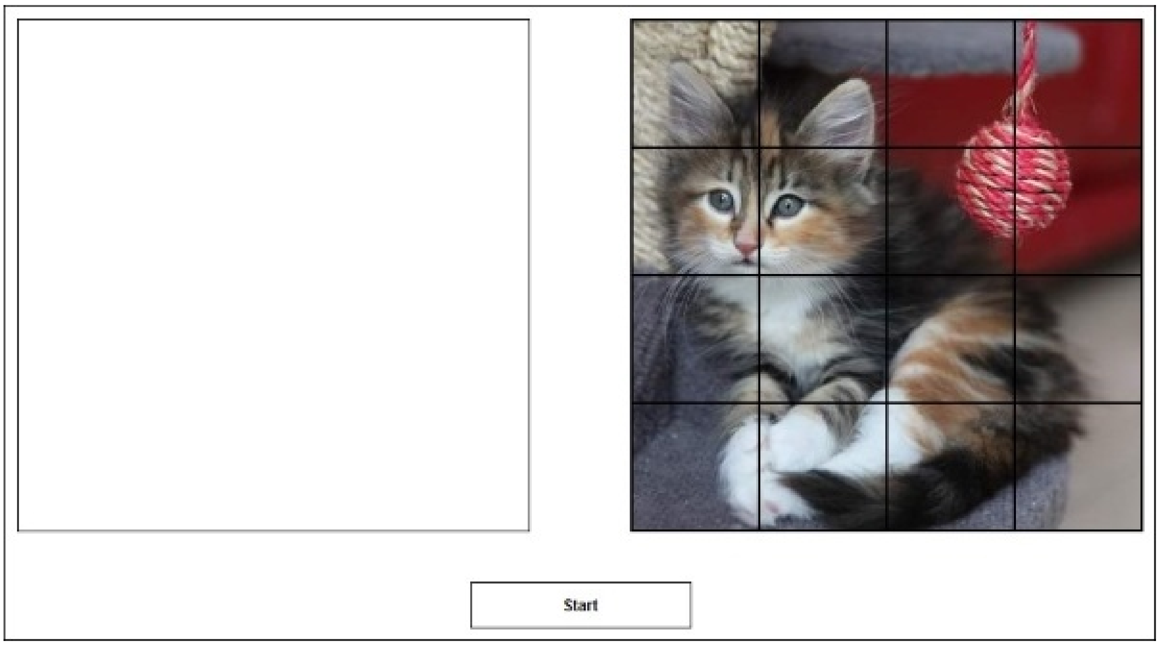
Write the following code in the `puzzle.js` file:

```
var rows = 4;
var cols = 4;
$(document).ready(function(){
  var sliceStr = createSlices(true);
  $('#puzzleContainer').html(sliceStr);
});

function createSlices(useImage){
  var str = '';
  var sliceArr = [];
  for(var i=0, top=0, c=0; i < rows; i++, top-=100)
  {
    for(var j=0, left=0; j<cols; j++, left-= 100, c++)
    {
      if(useImage)
      {
        sliceArr.push('<div style="background-position: ' + left + 'px ' +
top +'px;" class="img" data-sequence="'+c+'">');
      }
      else
      {
        sliceArr.push('<div style="background-image:none;" class="img
imgDroppable">');
      }
      sliceArr.push('</div>');
    }
  }
  return sliceArr.join('');
}
```

The 16 div elements will be in the form of a grid of 4 rows and 4 columns. In the preceding code, we defined two variables, `rows` and `cols`, and set their value to `4`.

Next, there is the `$(document).ready(function()` handler, in which we will write our code. Inside this handler, we call the `createSlices` function. This function will create the required 16 div elements and return a string with their HTML structure. This string will then be inserted into the `puzzleContainer` div element.

After you have written this code, save the `puzzle.js` file and refresh the `index.html` page on your browser. You will see a screen resembling the following screenshot:



Now let’s look at the `createSlices` function in detail.

We defined a variable named `str` to store the HTML structure. Next, there are two `for` loops. In the outer loop, we initialized another variable named `top` to `0`, which will be decremented by `100` in each iteration.

Similarly, inside the inner loop, another variable named `left` is defined, and this will also be decreased by `100` in each iteration. Inside the inner loop, a `div` element is created, where we set the div’s `left` and `top` values using the `background-position` CSS property. This is done in order to create all 16 slides with appropriate images.

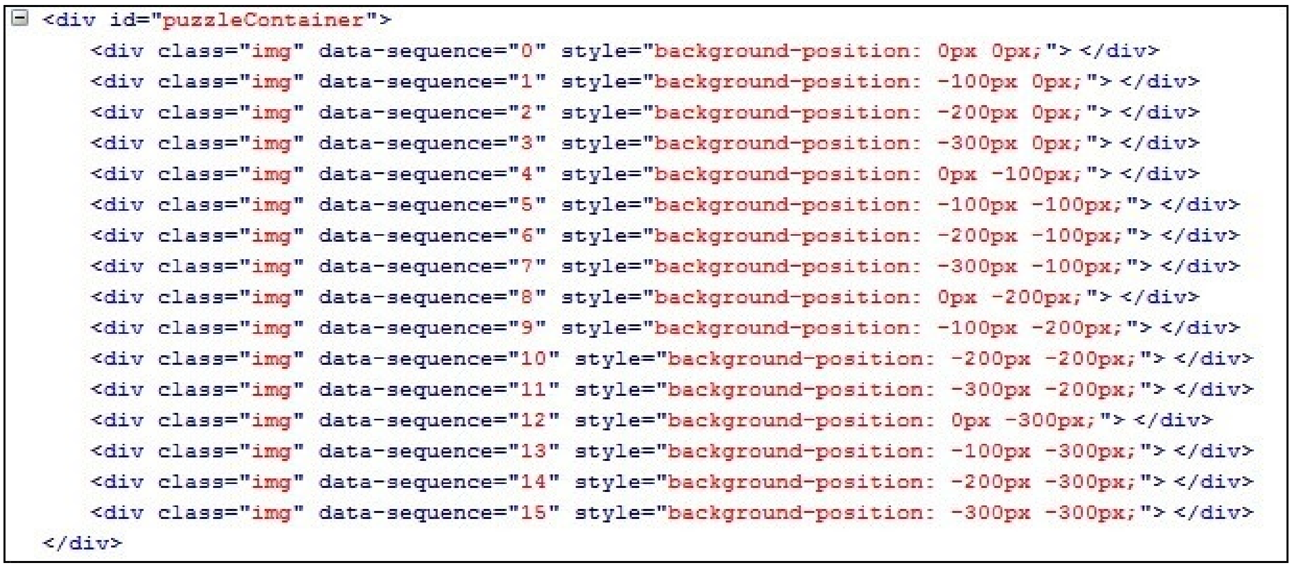
A CSS class named `img` is also added to the `div` element. We have already defined CSS properties for this class in the `index.html` file. This class sets the background image as `kitty.jpg` for the `div` element. It also defines the `height` and `width` of the `div` as `100 px` each, and a `border` of `1 px` is also applied.

A data attribute named `data-sequence` is also added to each div. This attribute will be used later to check whether all the div elements are arranged correctly or not. Its value will be `0` for the first div, `1` for the second div, `2` for the third div, and so on until `15`, which is set as a value for the last div. Once both the loops are completed, we return the complete DOM structure from the function. This structure will now be inserted in `div puzzleContainer`.

The CSS background-position property

To create a complete image using different pieces, we will need perfect placement of the `background-image` property. The `background-position` property defines the starting `left` and `top` positions of the background image for that specific div element. Therefore, if we define the background position as `background-position: 0px 0px`, it means that the image will get positioned at the top-left corner of element. Similarly, if we set `background-position: -100px 0px`, the left corner will skip the initial 100 pixels of the image.

To understand this more clearly, go to the browser page and inspect the DOM using Firebug (you can download this for Firefox from https://addons.mozilla.org/en-US/firefox/addon/firebug/) or Chrome DevTools (check out the help on Google Chrome DevTools at https://developer.chrome.com/devtools). You will see that the DOM structure resembles the following screenshot:



```
<div id="puzzleContainer">
    <div class="img" data-sequence="0" style="background-position: 0px 0px;"> </div>
    <div class="img" data-sequence="1" style="background-position: -100px 0px;"> </div>
    <div class="img" data-sequence="2" style="background-position: -200px 0px;"> </div>
    <div class="img" data-sequence="3" style="background-position: -300px 0px;"> </div>
    <div class="img" data-sequence="4" style="background-position: 0px -100px;"> </div>
    <div class="img" data-sequence="5" style="background-position: -100px -100px;"> </div>
    <div class="img" data-sequence="6" style="background-position: -200px -100px;"> </div>
    <div class="img" data-sequence="7" style="background-position: -300px -100px;"> </div>
    <div class="img" data-sequence="8" style="background-position: 0px -200px;"> </div>
    <div class="img" data-sequence="9" style="background-position: -100px -200px;"> </div>
    <div class="img" data-sequence="10" style="background-position: -200px -200px;"> </div>
    <div class="img" data-sequence="11" style="background-position: -300px -200px;"> </div>
    <div class="img" data-sequence="12" style="background-position: 0px -300px;"> </div>
    <div class="img" data-sequence="13" style="background-position: -100px -300px;"> </div>
    <div class="img" data-sequence="14" style="background-position: -200px -300px;"> </div>
    <div class="img" data-sequence="15" style="background-position: -300px -300px;"> </div>
</div>
```

This structure clearly shows 16 different divs, each having a different background-position setting. You can play with these values in **Firebug** or Chrome **Developer tools** in the options provided by the browser by increasing or decreasing their values to see how the background images are positioned on a puzzle piece.

Starting the game

Now that we have our puzzle pieces ready, we need to implement the **Start** button by adding an event handler for it. This event handler will shuffle all the slices created earlier and will place them at random positions in the div, having `pieceBox` as the `id`. The following code needs to be added to the `$(document).ready(function()` handler:

```
$('#start').on('click', function()
{
  var divs = $('#puzzleContainer > div');
  var allDivs = shuffle(divs);
  $('#pieceBox').empty();
  allDivs.each(function(){
    var leftDistance = Math.floor((Math.random()*280)) + 'px';
    var topDistance = Math.floor((Math.random()*280)) + 'px';
    $(this)
    .addClass('imgDraggable')
    .css({
      position : 'absolute',
      left : leftDistance,
      top : topDistance
    });
    $('#pieceBox').append($(this));
  });

  var sliceStr = createSlices(false);
  $('#puzzleContainer').html(sliceStr);

  $(this).hide();
  $('#reset').show();

});
```

Also, outside the `$(document).ready(function()` handler, define the `shuffle` function. This is the same function that we used in Chapter 1, *Designing a Simple Quiz Application*:

```
function shuffle(o)
{
  for(var j, x, i = o.length; i; j = Math.floor(Math.random() * i), x = o[-
-i], o[i] = o[j], o[j] = x);
  return o;
}
```

We registered a click event handler to the list item with the `start` ID. In the first line, we find all the `div` elements inside `puzzleContainer` in a `divs` variable. We pass this array to the `shuffle` function in the next line, which randomizes this array and returns it in a variable named `allDivs`. Now the `allDivs` variable is an array of puzzle pieces (`div` elements) in a random order. We need to place these pieces in the `piecebox` div.

Since we want these pieces to look scattered inside the `pieceBox` div, first we loop over the elements of the `allDivs` array. In each loop iteration, we generate two random numbers for the left and top positions for each div. We then set the div's position to absolute and add the left and top values. Since the `pieceBox` div has its position set to relative, each of these divs will be positioned using the left and top values relative to `pieceBox`. A `css` class, `imgDraggable`, is also added to each div. This class name will be used while dragging and dropping pieces. Finally, the div is appended to `pieceBox`.

The next line uses the `createSlices` function again to create a DOM with empty divs and without any background image. The DOM created using this function will be inserted to the `puzzleContainer` div again. Note that `false` is passed as a parameter to the `createSlices` function this time. This is because we do not want any background image in `puzzleContainer` when the game starts. This will require some modification in the `createSlices` function.

Modify the `createSlices` function written earlier to match the following code:

```
function createSlices(useImage)
{
  var str = '';
  for(var i=0, top=0, c=0; i < rows; i++, top-=100)
  {
    for(var j=0, left=0; j<cols; j++, left-= 100, c++)
    {
      if(useImage)
      {
        str+= '<div style="background-position: ' + left + 'px ' + top
+'px;" class="img" data-sequence="'+c+'">';
      }
      else
      {
        str+= '<div style="background-image:none;" class="img
imgDroppable">';
      }
      str+= '</div>';
    }
  }
  return str;
}
```

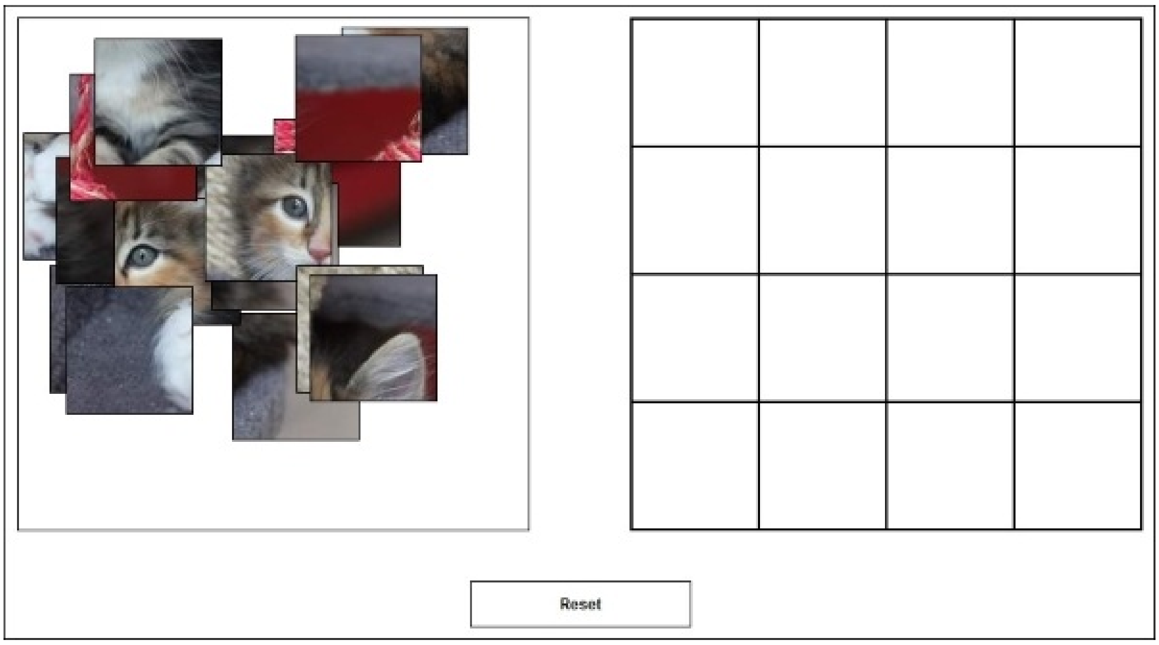
Note

Do not forget to change the function call in the first line inside the `$(document).ready(function())` section. Make sure `var sliceStr = createSlices();` is written as `var sliceStr = createSlices(true);`.

If the `useImage` argument for the `createSlices` function is set to `true`, the background image will be used. If it is `false`, no background image will be set but a class named `imgDroppable` will be added. This class will be used to attach event handlers to the places where the puzzle pieces will be dropped.

Finally, after preparing the DOM for the `pieceBox` and `puzzleContainer` divs, the **Start** button is hidden and the **Reset** button is displayed.

Reload the HTML page in your browser and you should see something resembling the following screenshot:



Reloading the page and clicking the **Start** button will display different positions of the puzzle pieces every time.

Handling events for puzzle pieces

To be able to move pieces and use the possible movements, we first need to add events. We will have to add two event handlers, one to make the puzzle pieces inside `pieceBox` draggable and second to make the `puzzleContainer` pieces droppable.

Inside the event handler of the **Start** button, add a new function call named `addPuzzleEvents()`.

Outside the `$(document).ready(function())` event handler, define the `addPuzzleEvents` function by writing the following code:

```
function addPuzzleEvents()
{
  $('.imgDraggable').draggable(
  {
    revert : 'invalid',
      start : function(event, ui ){
      var $this = $(this);
      if($this.hasClass('pieceDropped'))
      {
        $this.removeClass('pieceDropped');
        ($this.parent()).removeClass('piecePresent');
      }
    }
  });

  $('.imgDroppable').droppable({
    hoverClass: "ui-state-highlight",
    accept : function(draggable)
    {

      return !$(this).hasClass('piecePresent');
    },
    drop: function(event, ui) {
      var draggable = ui.draggable;
      var droppedOn = $(this);
      droppedOn.addClass('piecePresent');
      $(draggable).detach().addClass('pieceDropped').css({
        top: 0,
        left: 0,
        position:'relative'
      }).appendTo(droppedOn);

      checkIfPuzzleComplete();
    }
  });

}
```

There are two important points to be remembered here. Whenever a draggable puzzle piece is dropped on a droppable space, a CSS class named `pieceDropped` will be added to that draggable piece , which will indicate that the puzzle piece has been dropped. Another CSS class, `piecePresent`, will be added to the droppable space on which the piece is dropped. The presence of the `piecePresent` CSS class on a space will indicate that the space already has a piece dropped on it and we will disallow dropping any other draggable pieces on it.

All the puzzle pieces in `pieceBox` have a CSS class, `imgDraggable`, applied to them. We initialized the draggable component for all such pieces. While initializing, we provided two options for the draggable component. The first option is `revert`, which we set to `invalid`. As you may recall from Chapter 1, *Designing a Simple Quiz Application*, `invalid` means that a draggable piece will revert to its original position if it has not been dropped on any space. This also means when a piece is dropped inside puzzleContainer, you will not be able to place it back inside pieceBox.

Secondly, we added a `start` event handler to the piece. This event handler is called when the dragging begins. In the preceding code, we check whether the element being dragged has the `pieceDropped` class applied to it. If the `pieceDropped` class is not present on it, it means the piece is still inside `pieceBox` and has not been dropped in `puzzleContainer` yet.

If the `pieceDropped` class has been applied to the element, it means the puzzle piece was already dropped and it is being dragged inside `puzzleContainer` only. In this case, we want to allow the puzzle piece to be dropped onto other droppables spaces present inside `puzzleContainer`. Therefore, we remove the `pieceDropped` class from the draggable piece. In the next line, we also remove the `piecePresent` class from its parent droppable because we want the parent droppable to accept other draggable items.

Next, we will prepare the droppable space. In `puzzleContainer`, we have 16 different divs, which are used to accept the puzzle pieces. All of these have the `imgDroppable` CSS class applied to them. We initialize the droppable component using for all elements that have the `imgDroppable` class. While initializing, we provide three options, which are as follows:

- `hoverClass`: In this option, we can specify the name of any CSS class, and it will be applied to the droppable element when a draggable element will be over it. Note that the class name will only be applied when an accepted draggable element is over the droppable element. In the preceding code, we used the `ui-state-highlight` class, which is available by default in jQueryUI themes.
- `accept`: This option specifies which draggable elements can be dropped on to a droppable space. Either a jQuery selector or a function can be provided. We are using a function here to check whether the current droppable space already has a draggable element dropped in it or not. If the current droppable already has the `piecePresent` class, we return `false`, which means that the draggable element will not be allowed to drop on the current droppable space.
- `drop`: This event takes place once an accepted draggable element (described in the previous bullet point) is dropped onto a droppable space. Once the draggable is dropped, we add the `piecePresent` CSS class to the droppable. We also want the dragged puzzle piece to fit to the parent droppable completely. For this, we remove the draggable element from the DOM using jQuery’s detach method. Then we add a

CSS class, `pieceDropped`, to this droppable space. We set its `left` and `top` positions to `0` and `position` to `relative`. Finally, we append it to the parent droppable. The CSS properties specified with it fit it to its parent droppable.

After each drop, we call the `checkIfPuzzleComplete` function to check whether the puzzle has been solved.

Checking for puzzle completion

Every time a piece is dropped inside `puzzleContainer`, we will have to check whether all the pieces are in the correct order or not. To do this, we will create a function named `checkIfPuzzleComplete`. This function will be called from the drop event of the droppables. Define this function as shown in the following code:

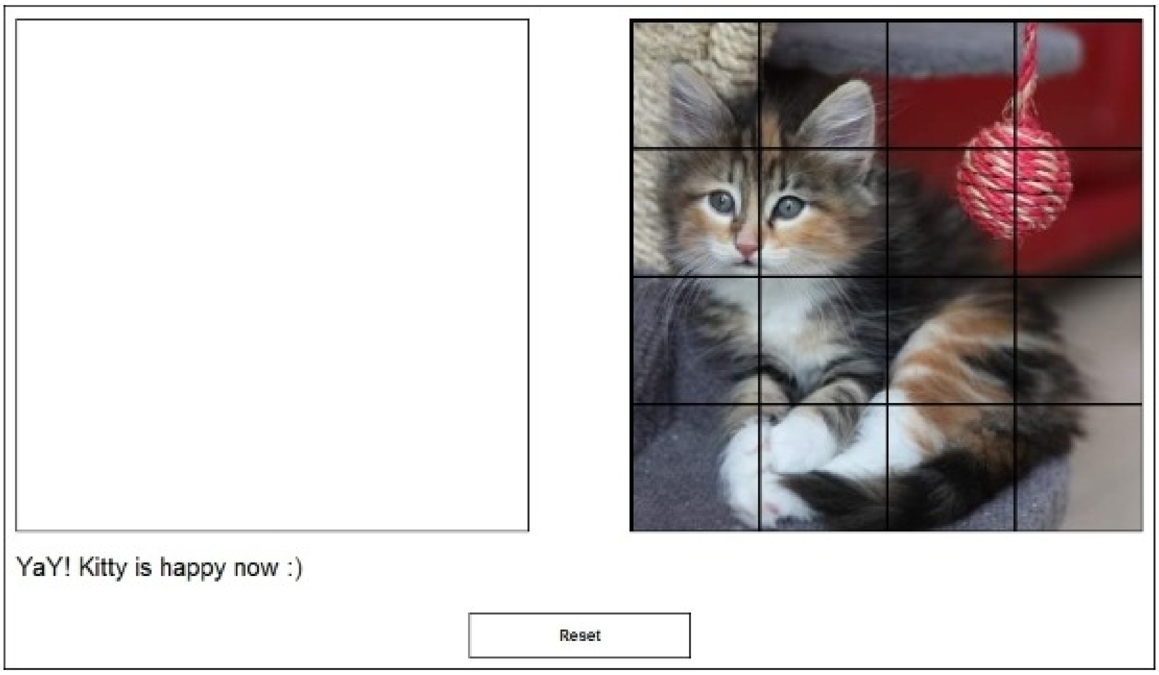
```
function checkIfPuzzleComplete()
{
  if($('#puzzleContainer div.pieceDropped').length != 16)
  {
    return false;
  }
  for(var i = 0; i < 16; i++)
  {
    var puzzlePiece = $('#puzzleContainer div.pieceDropped:eq('+i+')');
    var sequence = parseInt(puzzlePiece.data('sequence'), 10);
    if(i != sequence)
    {
      $('#message').text('Nope! You made the kitty sad :(').show();
      return false;
    }
  }
  $('#message').text('YaY! Kitty is happy now :)').show();
  return true;
}
```

It doesn't make any sense to check the puzzle if all 16 pieces have not been placed inside `puzzleContainer`. Since each puzzle piece dropped inside `puzzleContainer` will have a `pieceDropped` CSS class, we find out how many div elements with `pieceDropped` classes are present. If they are less than 16, we can assume that all pieces have not been placed inside the puzzle and return `false` from the function. If all 16 pieces are present inside `puzzleContainer`, we proceed to next step.

You may remember that we assigned a `data-sequence` attribute to each puzzle piece. In a correctly solved puzzle, all div elements will be in a sequence, which means their `data-sequence` attributes will have values from 0 to 15 in ascending order. Similarly, in an incorrectly solved puzzle, the `data-sequence` attributes of all `div` elements will still have values from 0 to 15, but they will not be in order.

The `for` loop checks the mentioned condition. We are running a loop from `0` to `15`. Each iteration of the loop picks a `div` element from `puzzleContainer` whose index is equal to current loop value. The `eq` jQuery function is used for this purpose. The sequence value for this `div` element is then retrieved and compared to the current loop value. If any of the values inside loop does not match this value, it will mean that the puzzle pieces are not in a sequence. In this case, we display the **Nope! You made the kitty sad :(** message inside the `div` with the `message` ID, and exit from the function.

If the loop completes all iterations, it means that all puzzle pieces are in order. Then we display the **YaY! Kitty is happy now :)** message and return from the function. A correctly solved puzzle will resemble the following screenshot:


YaY! Kitty is happy now :)

Resetting the puzzle

To reset the puzzle, all we need to do is create the pieces again and fill puzzleContainer with them. Write the following code inside the `$(document).ready(function())` handler to handle the reset button events:

```
$('#reset').on('click', function()
{
  var sliceStr = createSlices(true);
  $('#puzzleContainer').html(sliceStr);
  $('#pieceBox').empty();
  $('#message').empty().hide();
  $(this).hide();
  $('#start').show();
});
```

In the preceding code, we used the `createSlices` function with the `true` parameter and inserted the generated HTML inside `puzzleContainer`. Next, we emptied the `pieceBox`. The success or error message displayed earlier is also hidden. Finally, the **Reset** button is hidden and the **Start** button is displayed again.

Note

We do not need to call the `addPuzzleEvents` function to add drag and drop events. This is because the events were already attached to the DOM the first time the **Start** button was clicked.

Improving the puzzle

This puzzle can be made more interesting in a number of ways. Some ideas have been listed here. You are encouraged to add more features of your own as well:

- Allow removal of pieces from `puzzleContainer` to `pieceBox`
- Take a rectangular image where number of columns and rows are different
- Add a countdown timer to check how long the user took to finish the puzzle
- Display three to four images to users and allow them to select one image to be used with the puzzle

Summary

We made a somewhat complex example in this chapter, where you learned to use some important events of draggable and droppable components. Both of these are important components, and I encourage you to practice and try different variations as much as possible.

In the next chapter, we will look at some other equally useful components such as dialog, tooltip, accordion, and so on.

Chapter 3. Creating a Website Tour

By now, we have become familiar with the extremely useful draggable and droppable components of jQuery UI, both of which were used in the previous chapter. We will now explore some other components in this chapter that will prove to be a valuable asset in your web development toolbox. We will learn practical usage of the dialog, tooltip, and accordion components together.

If you change the layout of your website, it becomes difficult for regular users to navigate the new website initially. This might also mean a loss of visitors, if users find difficulty in searching for specific links or sections of the website they were familiar with earlier.

We will address this problem in this chapter and solve it by creating a website tour for a page. We will first design a simple home page for an imaginary company, and then create the tour to navigate different sections in the page. We will place a **Take a Tour** button on the page. Clicking on this button will start the tour. Each step of the tour will be a jQuery UI dialog box that will be positioned near the link or section we want to explain. The dialog box will have a title specific to the section and some text/HTML that will explain the functionality of the link or section. We will also place three buttons called **Previous**, **Next**, and **End Tour** to navigate the tour. While navigating with the **Previous** and **Next** buttons, it is possible that a section where the tour box has to be displayed is not in the current viewport. In such a case, we will scroll the page using jQuery. This will make sure that the tour box is visible in viewport.

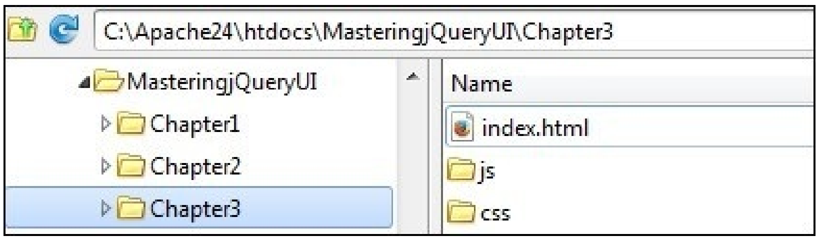
Apart from this, there will also be helpful tooltips in different parts of the page. We will use the following jQuery UI components to build the complete page with the tour feature:

- Dialog
- Buttons
- Tooltip
- Accordion
- Effects

Getting ready

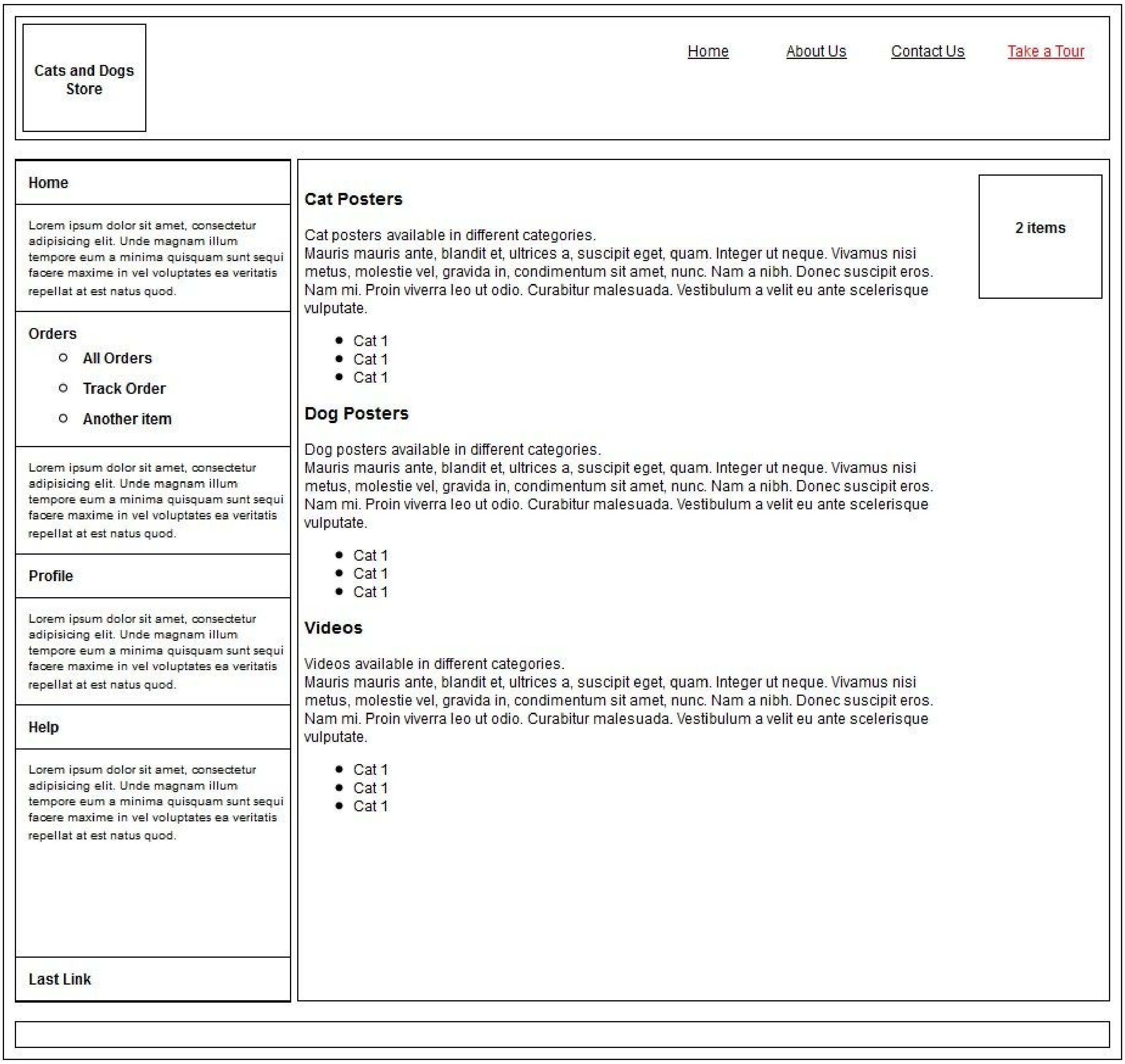
We will start by creating a new folder named `Chapter3` inside the `MasteringjQueryUI` folder. As explained in Chapter1, *Designing a Simple Quiz Application*, copy the `js` and `css` folders inside the `Chapter3` folder. Create a file and name it `index.html` inside the `Chapter3` folder, and create another JavaScript file named `tour.js` inside the `js` folder.

The file structure of the `Chapter3` folder should look like this now:



Designing the home page

Let’s assume our imaginary company is called *Cats and Dogs Store* and it sells goods for cats and dogs. For this company, we will design a page that will contain a header with a logo on the left and some links on the right. There will be a two-column layout underneath it. The left column will be a menu with several links and the right column will have an accordion and a “shopping cart” box. Each link or section where we want to display a step of the tour will be given an ID. These IDs will be used in JavaScript to make the tour functional. After the page is designed, it will look like the this:



Cats and Dogs Store

Home About Us Contact Us Take a Tour

Home

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Unde magnam illum tempore eum a minima quisquam sunt sequi facere maxime in vel voluptates ea veritatis repellat at est natus quod.

Orders
- All Orders
- Track Order
- Another item

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Unde magnam illum tempore eum a minima quisquam sunt sequi facere maxime in vel voluptates ea veritatis repellat at est natus quod.

Profile

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Unde magnam illum tempore eum a minima quisquam sunt sequi facere maxime in vel voluptates ea veritatis repellat at est natus quod.

Help

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Unde magnam illum tempore eum a minima quisquam sunt sequi facere maxime in vel voluptates ea veritatis repellat at est natus quod.

Last Link

Cat Posters

Cat posters available in different categories.
Mauris mauris ante, blandit et, ultrices a, suscipit eget, quam. Integer ut neque. Vivamus nisi metus, molestie vel, gravida in, condimentum sit amet, nunc. Nam a nibh. Donec suscipit eros. Nam mi. Proin viverra leo ut odio. Curabitur malesuada. Vestibulum a velit eu ante scelerisque vulputate.

- Cat 1
- Cat 1
- Cat 1

Dog Posters

Dog posters available in different categories.
Mauris mauris ante, blandit et, ultrices a, suscipit eget, quam. Integer ut neque. Vivamus nisi metus, molestie vel, gravida in, condimentum sit amet, nunc. Nam a nibh. Donec suscipit eros. Nam mi. Proin viverra leo ut odio. Curabitur malesuada. Vestibulum a velit eu ante scelerisque vulputate.

- Cat 1
- Cat 1
- Cat 1

Videos

Videos available in different categories.
Mauris mauris ante, blandit et, ultrices a, suscipit eget, quam. Integer ut neque. Vivamus nisi metus, molestie vel, gravida in, condimentum sit amet, nunc. Nam a nibh. Donec suscipit eros. Nam mi. Proin viverra leo ut odio. Curabitur malesuada. Vestibulum a velit eu ante scelerisque vulputate.

- Cat 1
- Cat 1
- Cat 1

2 items

Writing markup for the page

Now that we have a basic idea about what the home page should look like, let's start writing some markup now. Using Notepad++ or your favorite text editor, write the following HTML markup in the index.html file:

```
<html>
  <head>
    <meta charset="utf-8">
    <title>Creating a Website Tour</title>
    <link rel="stylesheet" href="css/ui-lightness/jquery-ui-
1.10.4.custom.min.css">
  </head>
  <body>
    <div id="dialog"></div>

    <div class="container">
    </div>
    <script src="js/jquery-1.10.2.js"></script>
    <script src="js/jquery-ui-1.10.4.custom.min.js"></script>
    <script src="js/tour.js" type="text/javascript"></script>
  </body>
</html>
```

The preceding code creates the skeleton of our page. In the head section, we linked the HTML page to the jQuery UI CSS file and just before closing the body tag, we included the jQuery source file, the jQuery UI JavaScript file, and the tour.js file.

Inside the body section, we defined an empty div with id dialog. This div element will be converted to dialog box using jQuery UI's dialog component to display the steps of the tour.

Next, we have a div with the container class that will wrap all the elements of the page. Inside the container class, we will first create the page header with the following HTML code:

```
<div class="header">
  <div id="logo">Cats and Dogs Store</div>
  <ul class="topLinks">
    <li>Home</li>
    <li>About Us</li>
    <li id="contact">Contact Us</li>
    <li id="startTour" title="Click to start Tour">Take a Tour</a></li>
  </ul>
</div>
<div class="clear">&nbsp;</div>
```

The header has two elements inside it. The first element is a div with an id logo and the second element is an unordered list ul with the class toplinks. This unordered list has four items inside it. The third list item has the id value contact. This list item will be used to show a tooltip with some contact details when the mouse hovers over it. The last list item with the id value startTour will act as a trigger button for starting the tour. After the header, there is an empty div with the class clear to clear the floats.

Now, we need to create two more `div` elements, one each for left and right columns, respectively. Write the following code after you have defined the header, as explained previously:

```
<div class="leftCol">
  <ul id="menu">

    <li>Home</li>
    <li><small>Lorem ipsum dolor sit amet, consectetur adipisicing elit.
Unde magnam illum tempore eum a minima quisquam sunt sequi facere maxime in
vel voluptates ea veritatis repellat at est natus quod.</small></li>

    <li id="orders">Orders
      <ul class="submenu">
        <li><a>All Orders</a></li>
        <li><a>Track Order</a></li>
        <li><a>Another item</a></li>
      </ul>
    </li>

    <li><small>Lorem ipsum dolor sit amet, consectetur adipisicing elit.
Unde magnam illum tempore eum a minima quisquam sunt sequi facere maxime in
vel voluptates ea veritatis repellat at est natus quod.</small></li>

    <li id="profile">Profile</li>

    <li><small>Lorem ipsum dolor sit amet, consectetur adipisicing elit.
Unde magnam illum tempore eum a minima quisquam sunt sequi facere maxime in
vel voluptates ea veritatis repellat at est natus quod.</small></li>

    <li id="help">Help</li>
    <li class="empty"><small>Lorem ipsum dolor sit amet, consectetur
adipisicing elit. Unde magnam illum tempore eum a minima quisquam sunt
sequi facere maxime in vel voluptates ea veritatis repellat at est natus
quod.</small></li>
    <li id="lastLink">Last Link</li>
  </ul>
</div>

<div class="rightCol">
  <div id="accordion">
    <h3 id="section1">Cat Posters</h3>
    <div>
      <p>
        Cat posters available in different categories.<br>
        Mauris mauris ante, blandit et, ultrices a, suscipit eget, quam.
Integer
        ut neque. Vivamus nisi metus, molestie vel, gravida in, condimentum
sit
        amet, nunc. Nam a nibh. Donec suscipit eros. Nam mi. Proin viverra
leo ut
        odio. Curabitur malesuada. Vestibulum a velit eu ante scelerisque
vulputate.
        <ul>
          <li>Cat 1</li>
          <li>Cat 1</li>
          <li>Cat 1</li>
        </ul>
      </p>
    </div>
    <h3 id="section2">Dog Posters</h3>
    <div>
      <p>
        Dog posters available in different categories.<br>
        Mauris mauris ante, blandit et, ultrices a, suscipit eget, quam.
Integer
        ut neque. Vivamus nisi metus, molestie vel, gravida in, condimentum
sit
        amet, nunc. Nam a nibh. Donec suscipit eros. Nam mi. Proin viverra
leo ut
        odio. Curabitur malesuada. Vestibulum a velit eu ante scelerisque
vulputate.
        <ul>
          <li>Cat 1</li>
          <li>Cat 1</li>
          <li>Cat 1</li>
        </ul>
      </p>
    </div>
    <h3 id="section3">Videos</h3>
    <div>
      <p>
        Videos available in different categories.<br>
        Mauris mauris ante, blandit et, ultrices a, suscipit eget, quam.
Integer
        ut neque. Vivamus nisi metus, molestie vel, gravida in, condimentum
sit
        amet, nunc. Nam a nibh. Donec suscipit eros. Nam mi. Proin
viverra leo ut
        odio. Curabitur malesuada. Vestibulum a velit eu ante scelerisque
vulputate.
        <ul>
          <li>Cat 1</li>
          <li>Cat 1</li>
          <li>Cat 1</li>
        </ul>
      </p>
    </div>

  </div>
  <div id="cart">2 items</div>
</div>
<div class="clear">&nbsp;</div>
```

First we created a `div` with class `leftCol`. We created an unordered list `ul` inside it that will act as a menu. This `ul` has some list items that will act as menu items but some are only placeholders with random text to make the page longer. Also, note that we assigned `id` values to some elements.

After `leftCol`, we created another `div` with the class `rightCol`. Inside it, there is yet another `div` with the `id` `accordion`. This `div` holds the markup that is required to create a jQuery UI accordion. Each panel of `accordion` consists of an `h3` element and a `div` element. `h3` will act as a header for that panel and `div` will become the body for that panel. Next to the `accordion` markup, there is another `div` that has the `id` `cart`.

This completes our HTML markup for the page and we are ready to assign CSS styling to the elements to make the page presentable.

Styling elements

Without any CSS styling to elements, HTML markup alone would make the page useless. To spice up the page, let's apply some CSS rules to decorate the page. In the `head` section, create a `<style>` block and write the following CSS rules for the different elements:

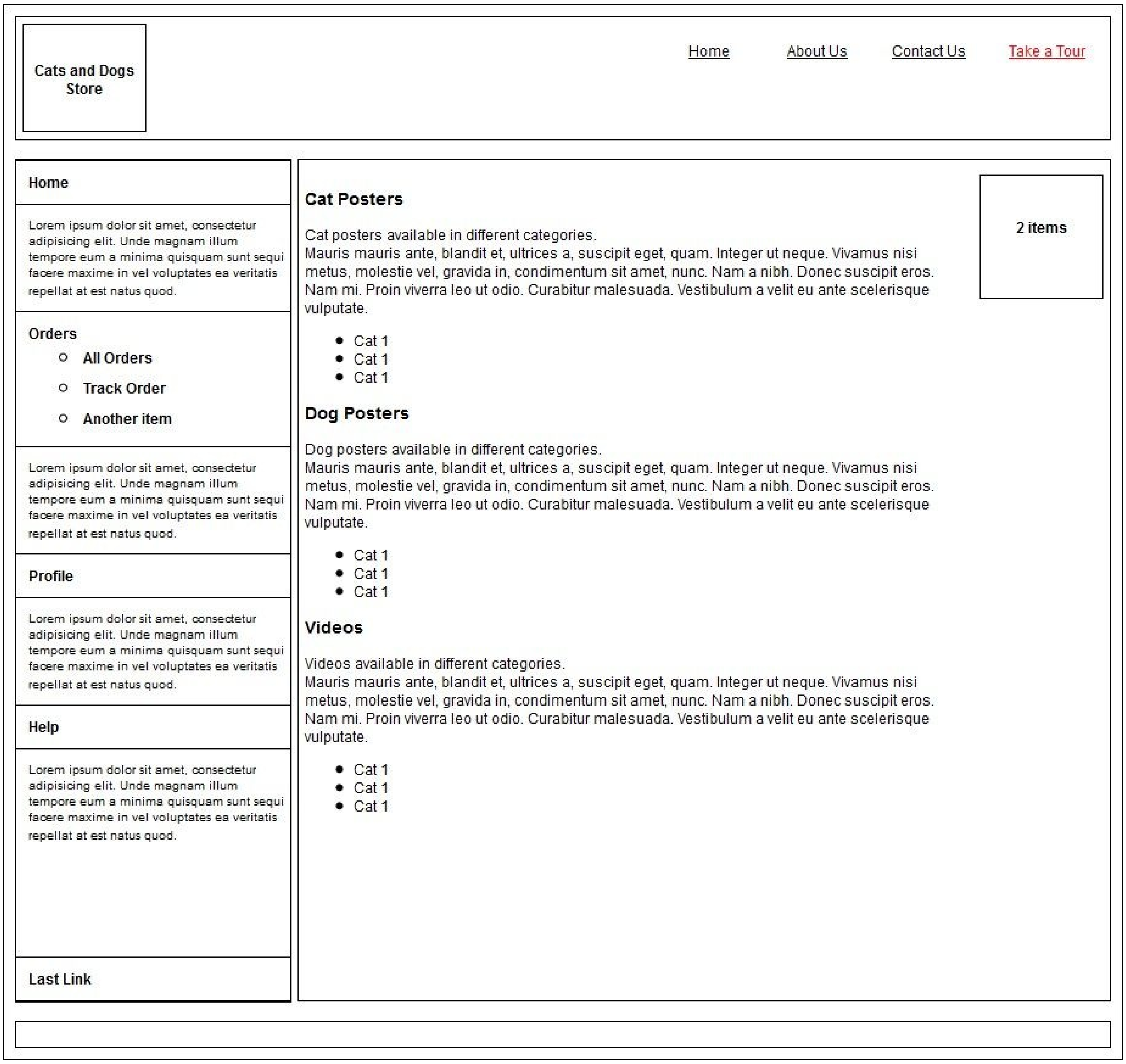
```
<style type="text/css">
  body
  {
    font-family:arial,verdana;
    font-size:12px;
    margin: 0px auto;
    width: 900px;
  }
  div.container
  {
    border: 1px solid #000;
    float:left;
    margin:10px auto 0;
    padding:10px;
    width: 100%;
  }
  .header
  {
    height: 100px; border: 1px solid;
  }

  #logo
  {
    border: 1px solid #000000;
    float: left;
    font-weight: bold;
    height: 57px;
    margin: 5px;
    padding-top: 30px;
    text-align: center;
    width: 100px;
  }

  ul.topLinks
  {
    float: right;
    list-style: none outside none;
    margin: 20px 20px 0 0;
    padding: 0;
    text-align: right;
    width: 70%;
  }
  ul.topLinks li
  {
    display: inline-block;
    margin: 0;
    padding: 0;
    text-decoration: underline;
    width: 15%;
    cursor:pointer;
  }
  #startTour
  {
    color: #ff0000;
  }

  .leftCol
  {
    border: 1px solid #000;
    float:left;
    min-height:500px;
    width:25%;
  }
  .rightCol
  {
    border: 1px solid #000;
    float: right;
    min-height:690px;
    width: 75%;
  }
  ul#menu {
    list-style:none;
    margin:0;
    padding:0;
  }

  ul#menu > li
  {
    padding:10px 5px 10px 10px;
    border-top: 1px solid #000;
    cursor:pointer;
    font-weight:bold;

  }
  ul#menu > li:last-child
  {
    border-bottom: 1px solid #000;
  }
  #accordion
  {
    width:80%;
    float:left;
    padding:10px 5px;
  }

  #cart
  {
      border: 1px solid #000000;
      float: right;
      font-weight: bold;
      height: 65px;
      margin-right: 5px;
      margin-top: 12px;
      padding-top: 35px;
      text-align: center;
```

```
      width: 100px;
  }
  a
  {
    text-decoration:none;
  }

  .submenu li
  {
    padding:5px;
  }
  small
  {
    font-weight:normal;
  }

  .empty
  {
    height:150px;
  }

  footer
  {
    border: 1px solid;
    padding: 10px 0px;
  }

  #dialog
  {
    display:none;
  }

  .clear
  {
    clear:both;
  }
</style>
```

The preceding CSS rules will change the layout and look of the elements in the page. The `div` with the `id` `logo` will become a box and will be placed `left` in the header. The `ul` list with the `class` `topLinks` will be floated to the `right`. Inside it, the `li` with `id` `tourStart` has been set to a red color so that it could stand out as an indicator to start the tour.

The `div` with the classes `leftCol` and `rightCol` has been made `25%` and `75%` wide, respectively, and a `border` has been applied to both of them. Similarly, all the `li` elements inside the `ul` list, the `leftCol` div has been padded and border has been applied to them. For elements inside `rightCol`, we floated the `div` with `id` `accordion` and `cart` to the `left` and `right`, respectively. We have not written any CSS for the `accordion` because its styling will be taken care of by jQuery UI's theme after the `accordion` is initialized.

After all the markup has been applied, run the `index.html` file in your local web server. You will see a home page similar to the following one:



Cats and Dogs Store

Home About Us Contact Us Take a Tour

Home

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Unde magnam illum tempore eum a minima quisquam sunt sequi facere maxime in vel voluptates ea veritatis repellat at est natus quod.

Orders

- All Orders
- Track Order
- Another item

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Unde magnam illum tempore eum a minima quisquam sunt sequi facere maxime in vel voluptates ea veritatis repellat at est natus quod.

Profile

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Unde magnam illum tempore eum a minima quisquam sunt sequi facere maxime in vel voluptates ea veritatis repellat at est natus quod.

Help

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Unde magnam illum tempore eum a minima quisquam sunt sequi facere maxime in vel voluptates ea veritatis repellat at est natus quod.

Last Link

Cat Posters

Cat posters available in different categories.
Mauris mauris ante, blandit et, ultrices a, suscipit eget, quam. Integer ut neque. Vivamus nisi metus, molestie vel, gravida in, condimentum sit amet, nunc. Nam a nibh. Donec suscipit eros. Nam mi. Proin viverra leo ut odio. Curabitur malesuada. Vestibulum a velit eu ante scelerisque vulputate.

- Cat 1
- Cat 1
- Cat 1

Dog Posters

Dog posters available in different categories.
Mauris mauris ante, blandit et, ultrices a, suscipit eget, quam. Integer ut neque. Vivamus nisi metus, molestie vel, gravida in, condimentum sit amet, nunc. Nam a nibh. Donec suscipit eros. Nam mi. Proin viverra leo ut odio. Curabitur malesuada. Vestibulum a velit eu ante scelerisque vulputate.

- Cat 1
- Cat 1
- Cat 1

Videos

Videos available in different categories.
Mauris mauris ante, blandit et, ultrices a, suscipit eget, quam. Integer ut neque. Vivamus nisi metus, molestie vel, gravida in, condimentum sit amet, nunc. Nam a nibh. Donec suscipit eros. Nam mi. Proin viverra leo ut odio. Curabitur malesuada. Vestibulum a velit eu ante scelerisque vulputate.

- Cat 1
- Cat 1
- Cat 1

2 items

Making the tour functional

Now that the layout of our page is complete, let’s discuss in brief how we are going to implement the tour functionality. We have already created a `tour.js` file that we will use for writing all the JavaScript code.

We will initialize the accordion for which we have already written the markup, and we will also initialize the tooltip that will appear on hovering over the **Contact Us** link.

After that, we will define a jQuery UI `dialog` component with some basic settings and buttons for navigating the tour. Next, we will define a JavaScript array that will contain all the steps of the tour. Finally, we will create a tour object, and we will write the functions that will initialize the tour, display a tour step, and handle the **Previous**, **Next,** and **End Tour** buttons.

Let’s write the code for the aforementioned steps and discuss them in detail.

Initializing accordion and tooltips

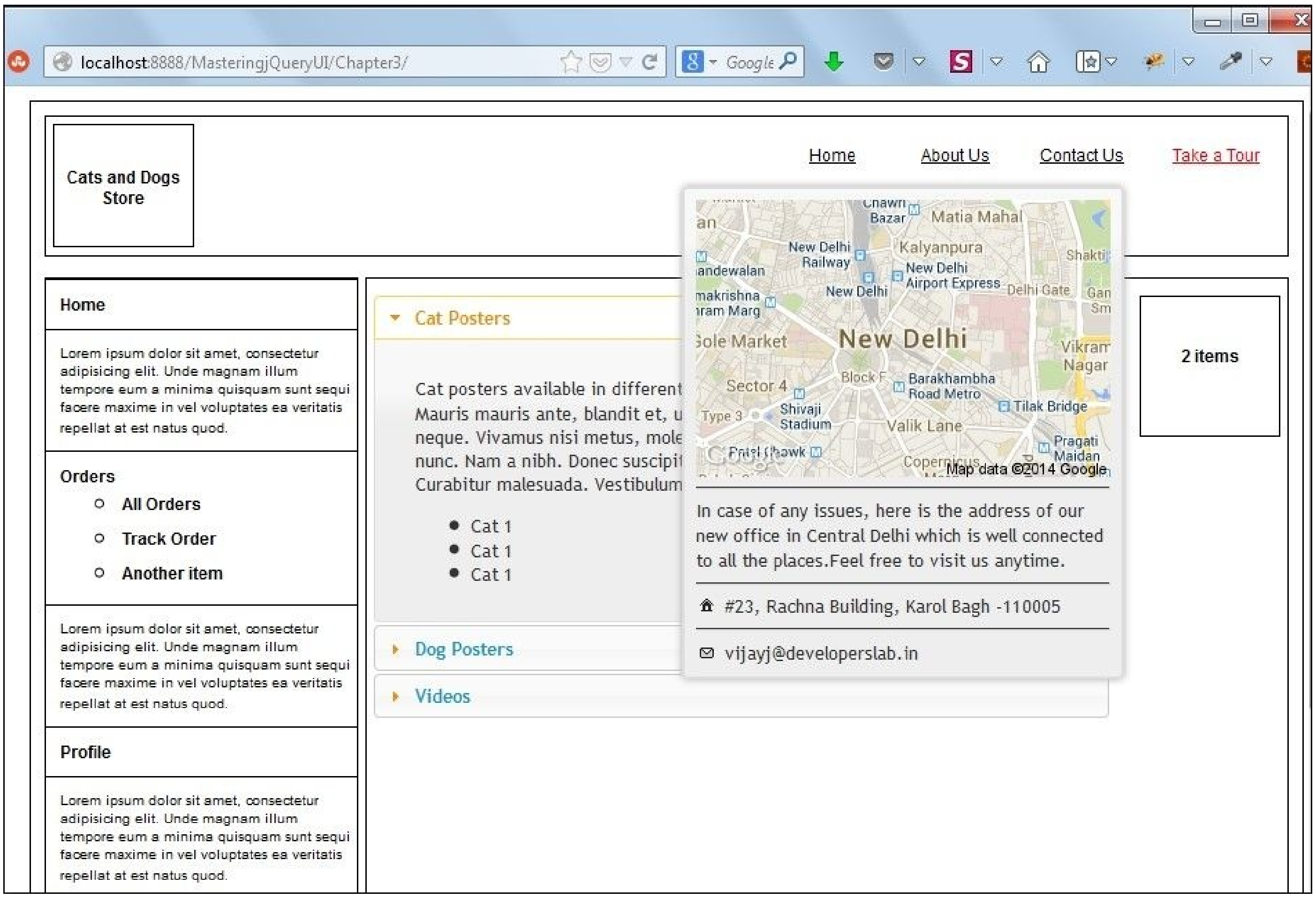
The best practice when writing JavaScript is to start by writing a jQuery `$(document).ready()` handler that ensures the related jQuery or JavaScript code is executed after the page has loaded. Open the `tour.js` file and start by writing the following code in it:

```
$(document).ready(function(){
  $('#accordion').accordion({animate : false});
  $(document).tooltip(
  {
    items : '#contact',
    content : function()
    {
      var strContact = '<img
src="http://maps.googleapis.com/maps/api/staticmap?
center=New+Delhi,India&zoom=13&size=300x200&sensor=false"/>';
      strContact+= '<hr/>In case of any issues, here is the address of our
new office in Central Delhi which is well connected to all the places.Feel
free to visit us anytime.';
      strContact+= '<hr><span class="ui-icon ui-icon-home" style="float:
left; margin-right: 5px;"></span>#23, Rachna Building, Karol Bagh -110005';
      strContact+= '<hr><span class="ui-icon ui-icon-mail-closed"
style="float: left; margin-right: 5px;"></span><a
href="mailto:awesomecompany@ourlocation.com">awesomecompany@ourlocation.com
</a>';
      return strContact;
    }
  });
});
```

In our markup, we assigned the `id` value `accordion` to the `div` that contains the markup for the `accordion`. Inside the `$(document).ready()` handler, the first line initializes the `accordion` with the `animate` option set to `false`. We also want a tooltip to appear on the **Contact Us** link. To achieve this, we initialize the `tooltip` component of jQuery UI.

If the `tooltip` component is initialized without providing any options, it displays the value of the `title` attribute in a tooltip. Since we want to display custom HTML, we have used two options, `items` and `content`, for it. The `items` option decides which elements will display the tooltip, and in the `content` option, we can provide any text or HTML to be displayed inside the tooltip. The value of `items` option has been set to `#contact`. To set the value of the `content` option, we created a `string`. This string contains an image and some information text. The source for image is a static image tile from Google Maps that shows the location of a place. You can set the value from center to any place of your choice.

Now, we can check the progress so far. Save the file and refresh the `index.html` page in the browser. You will find that jQuery UI's accordion has been styled using jQuery UI's theme and has also become active. Hovering your mouse over the **Contact Us** link will show the tooltip with an image and the text we defined in the `content` option while initializing the `tooltip`.



Lorem ipsum dolor sit amet, consectetur adipisicing elit. Unde magnam illum tempore eum a minima quisquam sunt sequi facere maxime in vel voluptates ea veritatis repellat at est natus quod.

Cat posters available in different… Mauris mauris ante, blandit et, u… neque. Vivamus nisi metus, mole… nunc. Nam a nibh. Donec suscipi… Curabitur malesuada. Vestibulum…

In case of any issues, here is the address of our new office in Central Delhi which is well connected to all the places.Feel free to visit us anytime.

#23, Rachna Building, Karol Bagh -110005

vijayj@developerslab.in

Defining the dialog

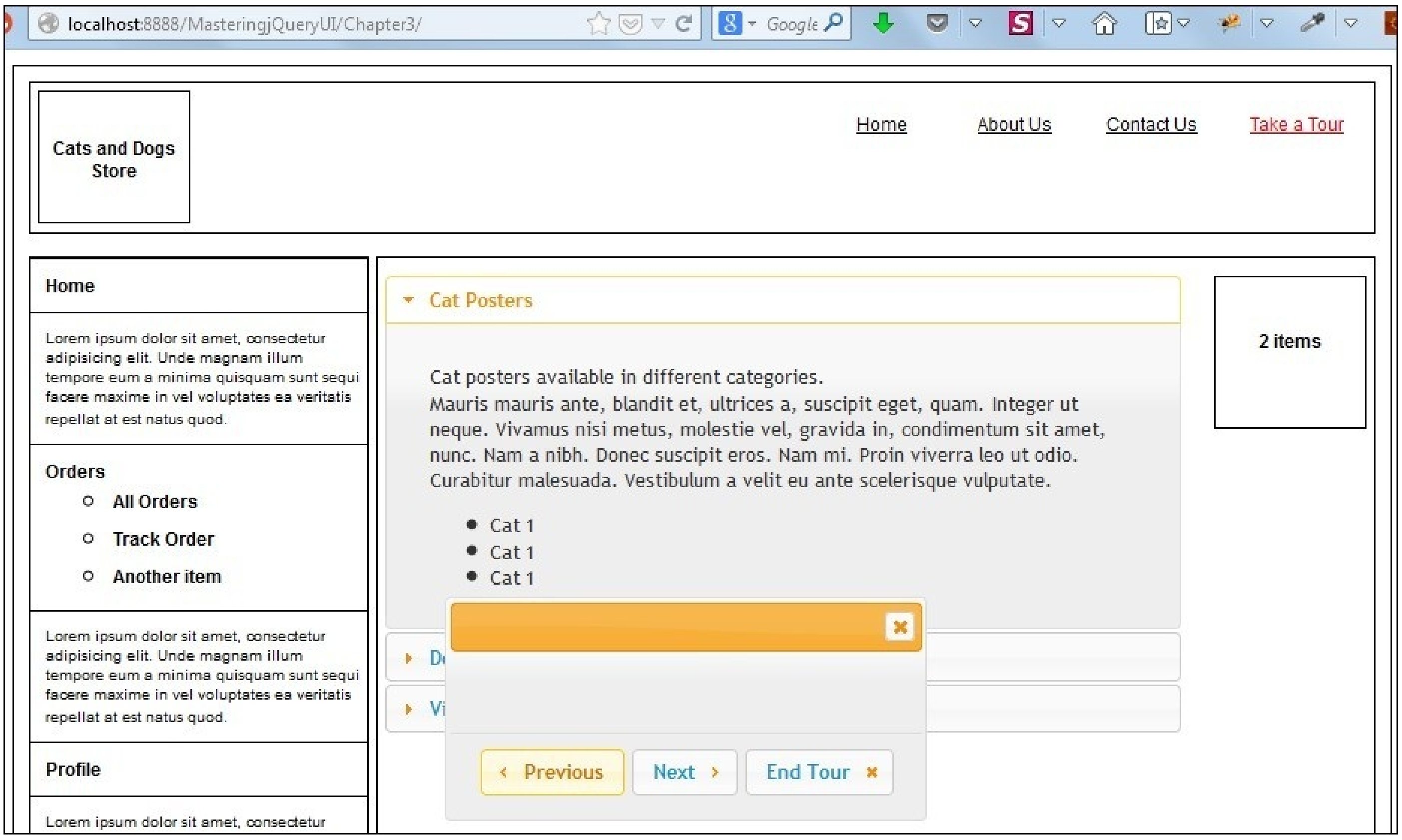
It’s time to initialize the dialog box. We already defined a `div` with `id` `dialog` in the markup. We will convert the said `div` to jQuery UI dialog box. For this, we need to define settings for the dialog and also the buttons that will appear in it. We will add the following code outside the `$(document).ready()` event handler to create the dialog:

```
var tourDialog = $('#dialog').dialog(
{
  minWidth : 315,
  draggable : false,
  buttons: [
  {
    id : 'buttonPrevious',
    text: 'Previous',
    click: function()
    {

    },
    icons:
    {
      primary: 'ui-icon-carat-1-w'
    }
  },
  {
    id : 'buttonNext',
    text: 'Next',
    click: function(event)
    {

    },
    icons:
    {
      secondary: 'ui-icon-carat-1-e'
    }
    },
  {
    text: 'End Tour',
    click: function()
    {

    },
    icons:
    {
      secondary: 'ui-icon-close'
    }

  }
  ],
  show : 'fold',
  hide : 'fold'
});
```

In the preceding code, we defined five options while creating the dialog. Let’s look at all of these one by one:

- `minWidth`: This option defines the minimum width the dialog must take. For our purposes, we have set it to `315`.
- `draggable`: Since the dialog box will appear at specific places denoting respective steps of the tour, we do not want users to drag the dialog. Hence, we have set this option to `false`.
- `buttons`: A dialog box can have one or more buttons that can be defined using the `buttons` option. We need to provide an array of objects for buttons. Each object in the array represents a button that will be displayed at the bottom of the dialog. We have defined three buttons, each having an `id`, a display `text`, a `click` event handler, and an icon. The first button has the `id` `buttonPrevious` and display `text` is **Previous**; we have also provided a primary icon for this button. The second button has `id` `buttonNext` and the display `text` is **Next**. Note that we have provided a secondary icon here. The difference between the primary and secondary icons is that the primary icon appears on the left of the text and the secondary icon appears to the right of the text. The third button has no id but has the display `text` **End Tour** and a secondary icon. We have left the click event handlers for all three buttons empty for now. We will go back to these event handlers later in the chapter.
- `show`: We can decide which effect will be used when the dialog is displayed. Any value mentioned in jQuery UI effects can be provided.
- `hide`: This is used to define the effect that will be used when the dialog closes.

After writing the preceding code, if you reload the `index.html` page in your browser, you will see that an empty dialog box with three buttons appears at the centre of the page as soon as the page finishes loading.



Lorem ipsum dolor sit amet, consectetur adipisicing elit. Unde magnam illum tempore eum a minima quisquam sunt sequi facere maxime in vel voluptates ea veritatis repellat at est natus quod.

Cat posters available in different categories.
Mauris mauris ante, blandit et, ultrices a, suscipit eget, quam. Integer ut neque. Vivamus nisi metus, molestie vel, gravida in, condimentum sit amet, nunc. Nam a nibh. Donec suscipit eros. Nam mi. Proin viverra leo ut odio. Curabitur malesuada. Vestibulum a velit eu ante scelerisque vulputate.

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Unde magnam illum tempore eum a minima quisquam sunt sequi facere maxime in vel voluptates ea veritatis repellat at est natus quod.

Lorem ipsum dolor sit amet, consectetur

We definitely do not want this to happen; the dialog must be displayed at the appropriate position only when the **Start Tour** button is clicked. Hence, we will add another option to the dialog, which will initialize the dialog but will not display it automatically. Add another option to the dialog initialization code with the following code:

```
autoOpen : false
```

Note

If you have other options defined after the preceding code, make sure that you end the line with a comma. However, there is no need for a comma if you defined `autoOpen` as the last option.

You can check again by reloading the page; the dialog box will not appear now.

Another important thing to note is that we used the variable `tourDialog` to keep a reference to the initialized dialog. This `tourDialog` variable will be used henceforth in the code to access the dialog.

Defining the tour steps

For the home page we have designed, our tour will have 12 steps. Each of these steps will be represented by a JavaScript object that has the following structure:

```
{
  element : '#logo',
  title : 'We have changed the logo !',
  content : 'Did you notice that we have made some changes to our logo as
well. ',
  sequence : 1
}
```

Let's see what each property does:

- `element`: This indicates the HTML element where the tour dialog box will appear
- `title`: This is the title that will be displayed in the dialog box
- `content`: This is the HTML content displayed in the dialog box
- `sequence`: This is a number indicating the step of the tour. We will start with `1` and proceed with 2, 3, and so on

Since we plan to display the tour steps on `accordion` as well, we will need two more properties:

- `isAccordion`: The value for this property will be set to `true` if the `element` property is part of an accordion.
- `accordionIndex`: This is the 0-based index of accordion panels where the tour step will be displayed. Using this property value, we will be able to open a specific panel of accordion before displaying a tour step.

Here is the full structure of steps of the tour for our example home page in this chapter:

```
var steps =
  [
    {
      element : '#logo',
      title : 'We have changed the logo !',
      content : 'Did you notice that we have made some changes to our logo
as well. ',
      sequence : 1
    },
    {
      element : '#menu',
      title : 'Menu On Left',
      content : 'We have placed all the menu items on left hand side for
quick access.',
      sequence : 2
    },
    {
      element : '#orders',
      title : 'Your Orders',
      content : 'Orders menu has a submenu which links to different
pages.',
      sequence : 3
    },
    {
      element : '#profile',
      title : 'Your Profile',
      content : 'This link will take you to your profile page where you
will be able to edit your profile and change password among other things',
      sequence : 4
    },
    {
      element : '#help',
      title : 'Get Help',
      content : 'Use this link to get help related to any issues',
      sequence : 5
    },
    {
      element : '#lastLink',
      title : 'Last Menu Link',
      content : 'This is last link of menu',
      sequence : 6
    },
    {
      element : '#section1',
      title : 'Buy Cat Posters',
      content : 'We have introduced a new category where you can buy
posters of cute cats ',
      isAccordion : true,
      accordionIndex : 0,
      sequence : 7
    },
    {
      element : '#section2',
      title : 'Buy Dog Posters',
      content : 'Dog lovers also welcome.',
      isAccordion : true,
      accordionIndex : 1,
      sequence : 8
    },
    {
      element : '#section3',
      title : 'Watch videos',
      content : 'We have collected some of the best videos from web and you
can see them here',
      isAccordion : true,
      accordionIndex : 2,
      sequence : 9
    },
    {
      element : '#cart',
      title : 'Your Cart',
      content : 'This is your shopping cart where all the products you have
selected will be displayed.',
      sequence : 10
    },
    {
      element : '#contact',
      title : 'Contact Us',
      content : function()
      {
        var strContact = '<img
src="http://maps.googleapis.com/maps/api/staticmap?
center=New+Delhi,India&zoom=13&size=280x200&sensor=false"/>';
        strContact+= '<hr/>In case of any issues, here is the address of
our new office in Central Delhi which is well connected to all the
places.Feel free to visit us anytime.';
        strContact+= '<hr><span class="ui-icon ui-icon-home" style="float:
left; margin-right: 5px;"></span>#23, Rachna Building, Karol Bagh -110005';
        strContact+= '<hr><span class="ui-icon ui-icon-mail-closed"
style="float: left; margin-right: 5px;">
</span>awesomecompany@ourlocation.com';
        strContact+= '<hr>You can take your mouse over Contact Us link if
you want to see this information later.';
        return strContact;
      },
      sequence : 11
    },
    {
      element : '#startTour',
      title : 'Thank You!',
      content : 'Thank you for going through through the tour.',
      sequence : 12
    }
  ];
```

We defined an array named `steps` with 12 objects. On going through each of these objects, you will see that we are starting with the `logo`, then proceeding to the menu, and so on. For quick reference, here are the `id` values of all the HTML elements where `tour` will be displayed one step at a time:

- `logo`
- `menu`
- `orders`
- `profile`
- `help`
- `lastLink`
- `section1`
- `section2`
- `section3`
- `cart`
- `contact`
- `startTour`

Initializing the tour

We have prepared the entire markup and JavaScript required to build the tour. Let's dive in and write some awesome JavaScript code. But before that, I would like to introduce you to a JavaScript best coding practice.

Since the code size will grow in this chapter, as well as in following chapters, we will use a JavaScript pattern called **object literal** to organize our code. Simply put, an object literal pattern implies defining a single JavaScript object with a set of comma-separated key-value pairs. Keys can be strings or identifiers and values can be strings, identifiers, or functions. The advantage of this pattern is that it does not pollute the global namespace. Keys that are defined inside the object will not be accessible directly outside that object. You will see this in practice in a moment.

First of all, visit the `$(document).ready()` event handler and add the following line of code in the end:

```
tour.init();
```

The preceding line calls the `init` function of a JavaScript object named `tour`. Outside the `$(document).ready()` event handler, let's define the `tour` object and its `init` function with the following code:

```
var tour =
  {
    triggerElement : '#startTour',
    tourStep : 0,
    tourSteps : steps,
    defaultTitle : 'Welcome to the tour !',
    defaultContent: 'This tour will show you the new changes we have made
to our site layout. <br> Please use next previous buttons to proceed. Click
the End Tour button whenever you want to finish the tour.',
    init : function()
    {
      if(this.tourSteps == undefined || this.tourSteps.length == 0)
      {
        alert('Cannot start tour');
        return;
      }

      $(this.triggerElement).on('click', function(event)
      {
        tour.showStep(tour.defaultTitle, tour.defaultContent, $(this));
        return false;
      });
    }
  };
```

The value of the `triggerElement` property is the `id` of the element, by clicking on which the tour will start. You can set it to any element. In our example page, we have created an element with the `id` `startTour` for this purpose. The `tourStep` property will keep track of the current step of the tour. Then, the `tourSteps` property to which we have assigned the variable `steps`. This variable is the array of twelve steps of the tour that we declared earlier. After this, there are two more properties named, `defaultTitle` and `defaultContent`. The `defaultTitle` property contains some text that will be displayed as the title of the dialog box when the `triggerElement` is clicked. Similarly, `tourContent` will serve as the content of dialog box.

The `init` property declares a closure. Inside this closure, we check to make sure whether the `tourSteps` variable has been defined and is actually an array. After this, a `click` event handler is added to the `triggerElement`. This event handler will be responsible for opening the dialog initially. From here, the user will start navigating the tour using the **Previous** and **Next** buttons. The event handler calls the `showStep` function. The `showStep` function will be responsible for displaying the dialog box and positioning it correctly next to the element, as defined in the `element` property of the `tourSteps` array. Three parameters are passed to the `showStep` function. Since this is the first time dialog opens, we pass `defaultTitle`, `defaultContent`, and the current trigger element `startElement`. We need to display the dialog box for the first time now. This will all be explained in the next section.

Displaying a tour step

The `showStep` function is responsible for displaying the dialog box and enabling or disabling the **Previous** and **Next** buttons. After the `init` function, write the following code to define the `showStep` function and another function named `prevNextButtons`:

```
showStep : function(tourStepTitle, tourStepContent, whichElement)
{
  this.prevNextButtons();
  $('body').animate(
  {
    scrollTop: $(whichElement).offset().top
  }, 500, function()
    {
      $('.ui-state-highlight').removeClass('ui-state-highlight');

      $(whichElement).addClass('ui-state-highlight');

      tourDialog.dialog('option', 'title', tourStepTitle);

      tourDialog.html(tourStepContent);

      tourDialog.dialog('option', 'position', { my: 'left top', at: 'right
top', of: whichElement, collision : 'flipfit' });

      tourDialog.dialog('open');
    });
},
prevNextButtons : function()
{
  $('#buttonNext').button('enable');
  $('#buttonPrevious').button('enable');
  if(this.tourStep == 0 || this.tourStep == 1)
  {
    $('#buttonPrevious').button('disable');
  }
  if(this.tourStep == this.tourSteps.length)
  {
    $('#buttonNext').button('disable');
  }
  return;
}
```

Note

If you keep the code for `showStep` after `init`, make sure that you have placed a comma (,) after the closing brace of the `init` closure.

Inside the `showStep` function, we receive three function arguments: the title text, content HTML, and the target element where a step of the tour will be displayed.

In the first line of `showStep`, we have called another function called `prevNextButtons`. Since the `showStep` function will display and position a tour step, we need to enable or disable the **Previous** and **Next** buttons. The `prevNextButtons` function is used to achieve this. If the user is on the first step of the tour, the **Previous** button will be disabled; if the user is on the last step of the tour, the **Next** button will be disabled.

First, we enable both the **Next** and **Previous** buttons. Then, we check the `tourStep` property to determine which step is being displayed currently. If it is the welcome dialog that is shown after clicking on the **Start Tour** link or the first step of the tour, we disable the **Previous** button. In the next line, we disable the **Next** button. If the value of the `tourStep` variable equals the length of all the steps in the array, it means the user is on the last step of the tour and hence we disable the **Next** button.

The **End Tour** button need not be handled, since the user can choose to end the tour during any step.

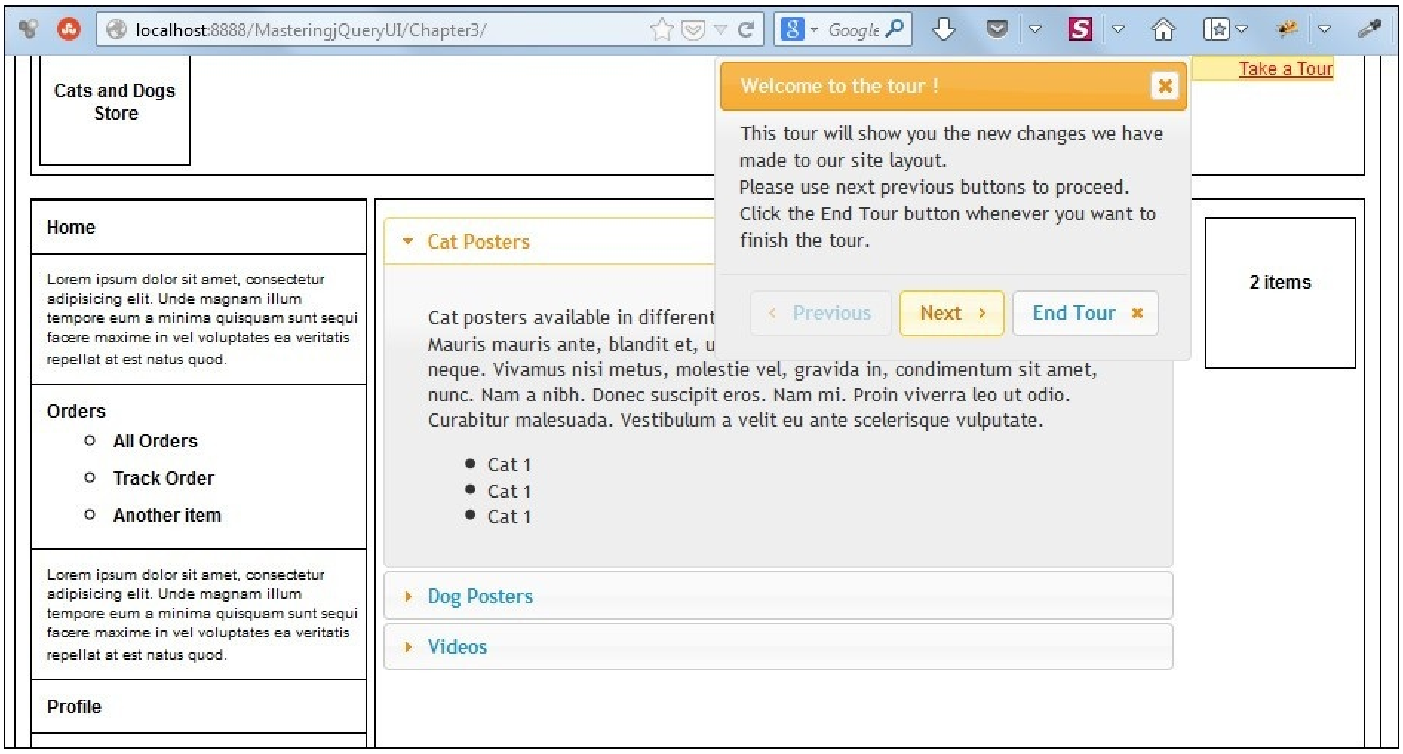
With the **Next** and **Previous** buttons taken care of, the control now returns to the `showStep` function.

Inside the `showStep` function, we used jQuery's `animate` function to scroll the page first so that the target element (`whichElement` received as a function argument) scrolls to the top of the viewport in 500 milliseconds. After scrolling is done, the callback function for `animate` fires. We remove the CSS class `ui-state-highlight` if it has been applied to any element. This is to ensure that there is no highlighted element in the page. In the very next line, the same class is applied to the current element (represented by `whichElement`) to highlight it. The `ui-state-highlight` class from jQuery UI's theme framework applies a yellowish background color to an element to make it look highlighted.

We have already initialized the `dialog` component inside the `$(document).ready()` event handler earlier. We now set the title of dialog box using jQuery UI's `option` method. Then, we set its inner HTML.

Now, we are left with one most important thing, that is, positioning the dialog. Remember that target elements can be in any corner of the web page, so we need to position the dialog so that its maximum area is available. To position the dialog, we have used the `position` option of jQuery UI. To properly position the dialog, we have set four properties of the `position` option. The name of the first property is `my`. We have set it to `left top`. Another property is `at`, which we have set to `right top`. Setting it to `right top` will align the dialog to the right top of the target element. The third property is `of`, where we provide the element that we want to position the dialog against. Since we want the dialog to be positioned against the target element, we provide it as a value of the `of` parameter. The last property is `collision`, which is used to ensure that the maximum part of the `dialog` is visible. We supplied the value `flip`. This property checks to either side of the element and positions it to the side where more space is available. With this, our positioning is done. We can now display the dialog.

In the last line of `showStep`, we called the `open` method of the jQuery UI `dialog` which displays the dialog. We can now check the result of our hard work so far in the browser. Reload the `index.html` page and click on the **Take a Tour** link. Windows will scroll a little and then the dialog will open with a `fold` effect. It will have the **Previous** button disabled.


localhost:8888/MasteringjQueryUI/Chapter3/

Cats and Dogs Store

Take a Tour

Welcome to the tour !

This tour will show you the new changes we have made to our site layout.
Please use next previous buttons to proceed.
Click the End Tour button whenever you want to finish the tour.

Previous | Next | End Tour

Home

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Unde magnam illum tempore eum a minima quisquam sunt sequi facere maxime in vel voluptates ea veritatis repellat at est natus quod.

Orders
- All Orders
- Track Order
- Another item

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Unde magnam illum tempore eum a minima quisquam sunt sequi facere maxime in vel voluptates ea veritatis repellat at est natus quod.

Profile

Cat Posters

Cat posters available in different Mauris mauris ante, blandit et, u neque. Vivamus nisi metus, molestie vel, gravida in, condimentum sit amet, nunc. Nam a nibh. Donec suscipit eros. Nam mi. Proin viverra leo ut odio. Curabitur malesuada. Vestibulum a velit eu ante scelerisque vulputate.

- Cat 1
- Cat 1
- Cat 1

Dog Posters

Videos

2 items

Making the Previous and Next buttons functional

We have successfully started the tour by showing a welcome dialog. Now, we need to make use of the **Previous** and **Next** buttons, and the `tourSteps` array to navigate them.

If you recall, while we created the dialog earlier in the chapter, we created the **Previous** and **Next** buttons and their event handlers. We will now make these buttons functional by calling appropriate code for moving forward or backward in the tour. Visit the code where buttons have been defined, and write the following code inside the `click` handler for the **Previous** button:

```
tour.navigate('previous');
```

Similarly, go to the `click` handler for the **Next** button and write the following code:

```
tour.navigate('next');
```

With this done, we will now define the `navigate` method. The `navigate` method will be defined inside the `tour` object after the code for the `prevNextButtons` function. Though you can define the function anywhere inside the `tour` object, writing the methods in sequence makes the code more readable:

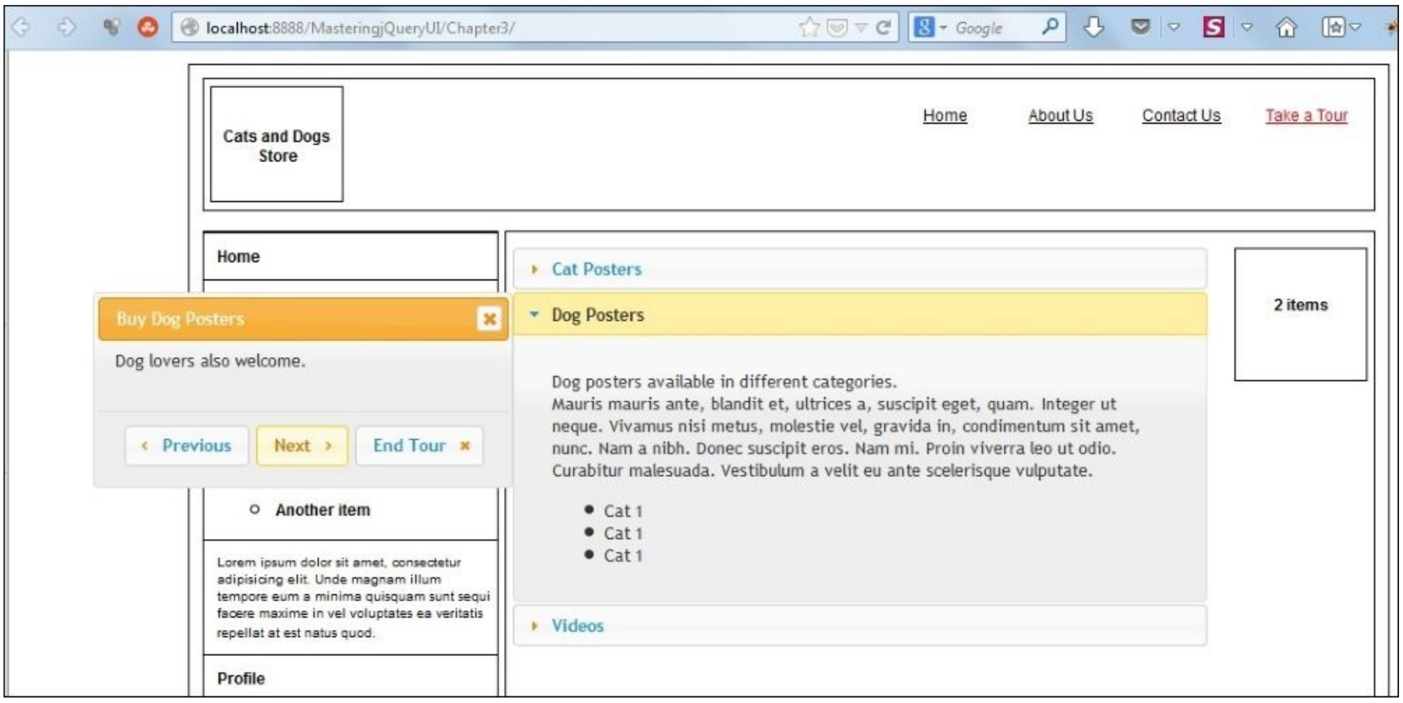
```
navigate : function(previousOrNext)
{
  if(previousOrNext == 'previous')
  {
    (this.tourStep) = (this.tourStep) - 1;
  }
  else
  {
    this.tourStep = this.tourStep + 1;
  }

  for(var i = 0; i<this.tourSteps.length; i++)
  {
    if(this.tourSteps[i].sequence == this.tourStep)
    {
      if(this.tourSteps[i].isAccordion)
      {
        $("#accordion" ).accordion("option", "active" ,
this.tourSteps[i].accordionIndex);
      }
      this.showStep(this.tourSteps[i].title, this.tourSteps[i].content,
this.tourSteps[i].element);
      return;
    }
  }
}
```

A single function will be used for handling both the **Previous** and **Next** buttons. The `navigate` function has an `previousOrNext` argument, based on which we can decide if the tour has to be moved forward or backward. If its value is `previous`, we increment the `tourStep` property of the `tour` object by one. Otherwise, we decrement its value by one.

The increment or decrement in value ensures that we will pick the correct `element`, `title`, and `content` properties from the `tourSteps` object while moving forward or backward.

A `for` loop is used to iterate in the `tourSteps` array. When we find that the `sequence` value of a `tourSteps` object matches the `tourStep` value, we call the `showStep` function and send the corresponding object's values for `element`, `title`, and `content`. There is another check that we place here for `accordion`. We have already defined the `isAccordion` and `accordionIndex` properties for the tour steps. So, if we find that the `isAccordion` value is `true`, we activate the corresponding `accordion` panel.

We can now check whether the **Previous** and **Next** buttons are working by reloading the page in our browser. A typical screen when the tour is on an `accordion` panel will look like this:



Dog lovers also welcome.

Dog posters available in different categories.
Mauris mauris ante, blandit et, ultrices a, suscipit eget, quam. Integer ut neque. Vivamus nisi metus, molestie vel, gravida in, condimentum sit amet, nunc. Nam a nibh. Donec suscipit eros. Nam mi. Proin viverra leo ut odio. Curabitur malesuada. Vestibulum a velit eu ante scelerisque vulputate.

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Unde magnam illum tempore eum a minima quisquam sunt sequi facere maxime in vel voluptates ea veritatis repellat at est natus quod.

Ending the tour

Ending the tour is very simple compared to the complex code we have written so far. Just visit the event handler for the **End Tour** button and use the following code:

```
tour.endTour();
```

We called the `endTour` method of the `tour` object. Let's define it as well. Add the following code inside the `tour` object:

```
endTour : function()
{
  this.tourStep = 1;
  $('.ui-state-highlight').removeClass('ui-state-highlight');
  tourDialog.dialog( 'close' );
}
```

The preceding code simply resets the `tourStep` to `1` so that the correct data is displayed when the tour is started again. The CSS class `ui-state-highlight` is also removed from any elements in the page.

Finally, the `close` method of dialog component is called, which hides the tour dialog with the fold animation.

Improving the tour

You can add some more interesting features to the tour to make it more dynamic. Here are some tips to get you started:

- Modify the tour so that it moves to the next step automatically after 5 seconds
- Ability to pause the automated tour
- Embed videos in the tour dialog box

Summary

In this chapter, you learned how components can be used in different and interesting ways. You went through the tooltip, the accordion, and most importantly the dialog components. You also learned to better organize our code using Object Literal pattern.

In the next chapter, we will move one step ahead and use AJAX and the reddit API to build a cool news reader.

Chapter 4. Creating a Tabbed News Reader

jQuery UI offers a number of components that can be used in a variety of combinations to create rich interfaces. We saw such a combination in the previous chapter, Chapter 3, *Creating a Website Tour*, where we used dialog, buttons, tooltips, and other components to create a neat website tour feature.

Moving forward, let's talk about tabs. Tabs are a very useful widget for the presentation layer, that is, the **user interface** (**UI**). Most common use cases for tabbed interfaces are UIs where there is limited space. Tabs have multiple panels in which different contents can be displayed. Clicking on a tab opens its corresponding panel.

In this chapter, you will focus on tabs and dialog components and learn to integrate the **reddit** API with the tabs. If you are not familiar with reddit, it is a news aggregation website where all the content is submitted by the users. There are different "subreddits" created by users for different topics such as web development, world news, technology, PHP, jQuery, and so on. Whenever content (text or a link to any website) is submitted by a user, other users can comment on it and upvote or downvote the content as well as other comments. This makes the quality content more visible and features on top of the list. Reddit also provides an API, using which we can retrieve the posts of a particular subreddit and the comments users made on a particular post.

Using the reddit API, we will create a tabbed interface as an exercise that we will call `MyjqReddit`. In the first tab, we will ask the name of a subreddit in a textbox and then we will load the posts of that subreddit in a new tab that will be created dynamically. For each post, we will create a **View comments** button. On clicking this button, we will use the API to fetch comments for that post. These comments will then be displayed in a dialog box. Users will also be able to close any tab by clicking on the close icon placed beside the tab name.

In this chapter, we will cover these two components:

- Tabs
- Dialog

Creating the folder structure

The first step is to create the required folder structure. We will follow the same format we have been using in previous chapters:

1. Create a folder named `Chapter4` inside the `MasteringjQueryUI` folder.
2. Inside the `Chapter4` folder, create an HTML file with the name `index.html`, which will contain our HTML markup.
3. Also, copy the `js` and `css` folders inside the `Chapter4` folder.
4. Now, inside the `js` folder, create a new file, and name it `myReddit.js`. This file will contain all the code required for creating our news reader.

Once this setup is complete, we can move to the next step – designing the page.

Designing the page

The page we are going to design will consist of a tab widget and markup for a dialog. Initially when the page loads, we will display one tab by default. This tab will serve as the home page of our news reader. The tab will have two HTML controls, a textbox and a button. All other tabs will be generated dynamically. Unlike other tabs, the first tab will be available at all times and users will not be able to remove or close it from the tabs panel. A dialog is required because we will use it to display comments for a particular post.

Writing markup for tabs

To create both the tabs and the dialog, we will follow the markup required by the jQuery UI library. Add the following code to our newly created `index.html` file:

```
<html>
  <head>
    <meta charset="utf-8">
    <title>MyjqReddit</title>
    <link rel="stylesheet" href="css/ui-lightness/jquery-ui-
1.10.4.custom.min.css">
    <style type="text/css"></style>
  </head>
  <body>
    <div id="redditTabs">
      <ul id="tabList">
        <li><a href="#tabs-1">Welcome</a></li>
      </ul>
      <div id="tabs-1" class="tabContent">
        <h2>Welcome to MyjqReddit</h2>
        <p>
          Please enter the name of any subreddit in textbox and click the
<strong>Add It</strong> button. The subreddit contents will be loaded in a
new tab.
          <br/>
          You can click the comments for a post by clicking the
<strong>"View Comments"</strong> link under the post title.
        </p>

        <label for="subredditName">Enter a subreddit name : </label>
        <input type="text" id="subredditName" name="subredditName" value=""
class="ui-widget-content ui-corner-all">
        <button id="addNewSubreddit" type="button">Add it</button>
        <hr/>
        <strong>To start with, you can try any of these subreddit names :
</strong><small>webdev, technology, AskReddit, IAMA, india,
worldnews</small>
        <hr/>
        <div class="ui-widget" id="errorContainer" style="display:none;">
          <div class="ui-state-error ui-corner-all">
            <span class="ui-icon ui-icon-alert"></span>
            <div id="errorMessage"></div>
          </div>
        </div>

      </div>
    </div>

    <div id="dialog">
      <div id="loader">
        Loading Comments… Please wait…
      </div>
      <div id="commentsList"></div>
    </div>
    <script src="js/jquery-1.10.2.js"></script>
    <script src="js/jquery-ui-1.10.4.custom.min.js"></script>
    <script src="js/myReddit.js" type="text/javascript"></script>
  </body>
</html>
```

We started by creating an HTML document and included the jQuery UI CSS file in the `head` section. Inside the `body` section, we created a `div` and assigned `id` value `redditTabs` to it. This `div` will act as a container for the tabs. Inside it, there is an unordered list `ul` with the `id` value `tabList`. Individual list items inside this list will act as tab headers. By default, we created one list item inside it. This list item contains an anchor that points to the `div` element with `id` value `tabs-1`.

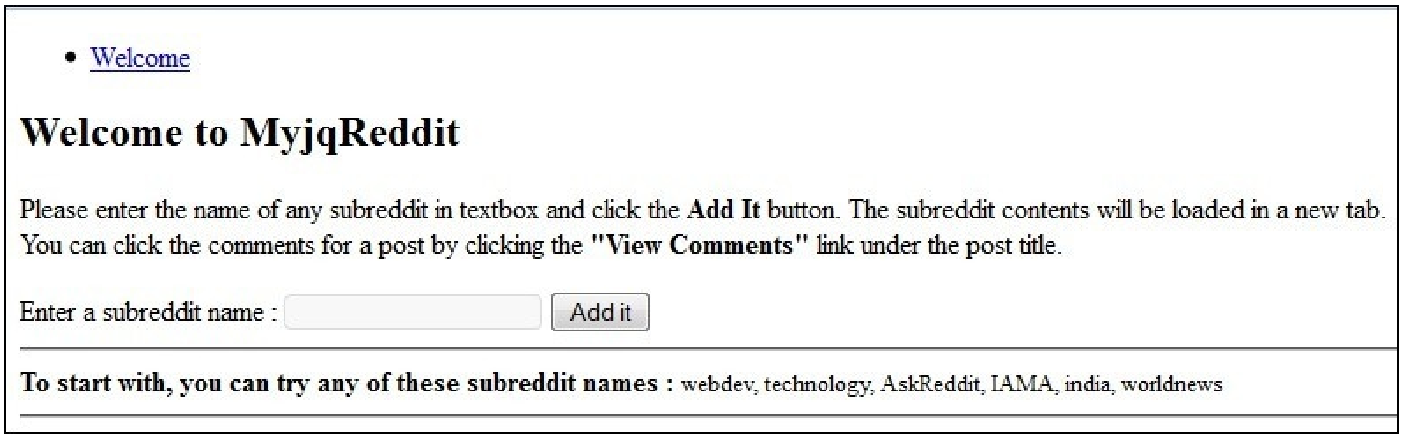
After creating the tab headers, tab panels need to be created for each of the defined tab headers. Since we declared only one tab header, we will define only one panel. This has been done by creating another `div` with `id` `tabs-1`. Note that we provided a link to this `id` inside the tab header we created before it. This is needed to let jQuery UI know which tab headers have to be linked to which panels.

Inside the tab panel `tabs-1`, we wrote some text to explain to the user what the page is about. There is also a textbox with `id` value `subredditName` and next to it there is a button with `id` vlaue `addNewSubreddit`.

Below these elements, we created a `div` with `id` `errorContainer` by utilizing the CSS classes provided by jQuery UI. We also set its `display` style to `none` as this `div` will be displayed only when there is any error. We used CSS `class` named `ui-state-error` on a `div` and inside it we have placed an icon and another `div` where any error message will be displayed.

After the `div` `redditTabs`, we defined a `div` with `id` value `dialog`. It will be used to display the comments.

Since we created the basic page layout, we can check it as well. Fire up your browser and open the `index.html` file in it. You should see a page similar to the following screenshot:



- Welcome

Welcome to MyjqReddit

Please enter the name of any subreddit in textbox and click the **Add It** button. The subreddit contents will be loaded in a new tab. You can click the comments for a post by clicking the **"View Comments"** link under the post title.

Enter a subreddit name : [ ] Add it

To start with, you can try any of these subreddit names : webdev, technology, AskReddit, IAMA, india, worldnews

Do not worry if the page looks raw. We are going to take care of basic styling in the next section.

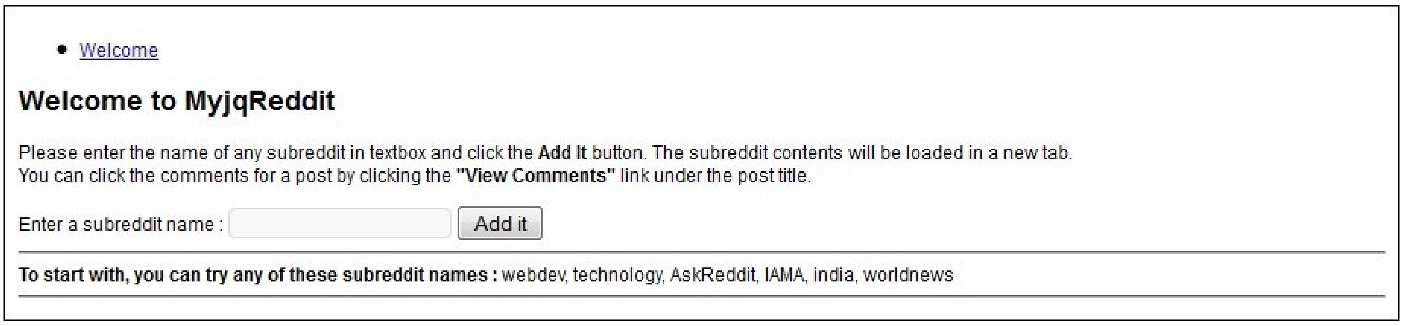
Styling the content

jQuery UI will apply its own style sheet of the downloaded theme once tabs are initialized. But before that, we need to add some styles of our own to make the look and feel consistent. The rules we are going to add will control the display of reddit posts in tabs, and the comments that will be displayed in the dialog. Here are the CSS rules that we will add inside the style part of the `head` section:

```
body{
  font-family:arial,verdana;
  font-size:12px;
  margin: 0px auto;
  width: 900px;
}
div#redditTabs{
  border: 1px solid #000;
  float:left;
  margin:10px auto 0;
  padding:10px;
  width: 100%;
}

a{
  text-decoration:underline;
}

.ui-tabs-panel .ui-icon
{
  float:left;
}

.extras {
  clear: both;
  padding-bottom: 20px;
}

.extras a {
  border: 1px solid #A2A2A2;
  cursor: pointer;
  display: block;
  float: left;
  margin-left: 5px;
  text-decoration: none;
  width: 140px;
}

.postTitle{
  display: block;
  padding: 5px 0px 5px 5px;
  font-weight:bold;
}

.postTitle a {
  color:#07c;
  text-decoration:none;
}

.postList{
  list-style: none;
  margin: 0;
  padding: 0 0 0 10px;
  width: 100%;
}

.postList li {
  background:none #fff;
  margin:5px 0;
  padding:5px;
}

.postDescription{
  border-top:1px solid #ddd;
  margin-top:5px;
  padding:5px 0 5px 5px;
  display:none;
}

.postDescription a {
  color:#07c;
}

.ui-icon-close{
  float:left;
  margin-top:5px;cursor:pointer;
}

small {
  font-size:12px;
}

.comments {
  background-color: #eee;
  list-style: none outside none;
  margin-bottom: 5px;
  margin-right: 0;
  margin-top: 5px;
  padding: 2px 0;
}

.comments a.username {
  color: #ff0000;
}

.comment {
  background-color: #FFFFFF;
  border-color: #5D5D5D;
  border-image: none;
  border-style: dotted dotted none;
  border-width: 1px 1px 0;
  margin-bottom: 10px;
  padding: 0 10px 2px;
}

.hide {
  display:none;
}

#dialog {
  display:none;
}


.clear {
  clear:both;
}
```

The CSS rules which we have defined, first add some default properties for the `body`. Then, we added CSS properties to customize the display of reddit posts and the comments that will be displayed in the dialog. We added these properties to make sure that the UI remains clutter-free and readable.

After adding these rules, if you reload the `index.html` page in your browser, it will look a bit more organized and readable.



- Welcome

Welcome to MyjqReddit

Please enter the name of any subreddit in textbox and click the **Add It** button. The subreddit contents will be loaded in a new tab.
You can click the comments for a post by clicking the **"View Comments"** link under the post title.

Enter a subreddit name : [ ] Add it

To start with, you can try any of these subreddit names : webdev, technology, AskReddit, IAMA, india, worldnews

After jQuery UI tabs and dialog components are initialized in the next section, the jQuery UI theme CSS will also be applied and the page will look much prettier.

Getting the code structure ready

As we did in previous chapter, Chapter 3, *Creating a Website Tour*, we will use the object literal pattern to keep our code manageable and inside a single object. We will define an object named `myReddit`, inside which there will be properties and methods (they can be called members as well) to create our news reader. Let’s start by writing the code for the same in the `myReddit.js` file:

```
var myReddit =
{
  apiURL : 'http://www.reddit.com',
  tabCount : 1,
  init : function()
  {
  },
  getJSONFromAPI : function(type, id)
  {
  },
  createTab : function(subredditName, postList)
  {
  },
  getPostListingHtml : function(postListing)
  {
  },
  displayComments : function(data)
  {
  },
  getCommentsHTML : function(commentsList)
  {
  },
  htmlDecode : function(input)
  {
  }
};

$(document).ready(function()
{
  myReddit.init();
});
```

We created an object called `myReddit` with some members. Let’s analyze these members one by one:

- The first name value pair is `apiURL`. This is the base URL for requests that will be made to reddit’s JSONP API. Since there will be separate URLs for posts and comments, we have only defined base URL which we will change depending on the request that will be sent.
- Next is `tabCount` which will be used to manage the addition and removal of tabs dynamically. Initially `tabCount` is set to `1` as we already have one tab present.
- Then, there are the methods by which we will implement the required functionality. We begin by defining an `init` function where we will do the basic initializations and add the required event handlers.

- The `getJSONFromAPI` method is responsible for getting responses from the reddit API by sending appropriate requests. We will use this method to retrieve the posts as well as retrieve the comments for a post.
- Once we have fetched the posts for a subreddit from the API, we will need to create a new tab and add it to existing tab structure. We will also have to create the DOM to display the posts in the newly created tab.
- As the name suggests, the `createTab` method will be used for this. It will also use the `getPostListingHtml` method to create the DOM using the API response. The `getPostListingHtml` method will take the API response as input and will create the DOM. It will return the created HTML structure back to the `createTab` method where it will be inserted into the DOM.
- Now that posts are taken care of, we defined another method named `displayComments`. This method will use the API response for comments (which we will fetch using the `getJSONFromAPI` method defined earlier) to create the DOM for displaying comments. It will use the `getCommentsHTML` method to iterate into the comments and create the DOM.
- Since comments could be nested multiple levels deep, the `getCommentsHTML` function will be called recursively to display all available comments.
- The helper method `htmlDecode` is used to unescape the HTML in the responses from reddit API.
- Finally, after the `myReddit` object definition, there is jQuery's document ready event handler where we call the `init` function of `myReddit`. Let's begin by defining the `init` method.

Adding event handlers in the init method

The `init` method is the first method of the `myReddit` object that is called. Since this is a starting point for us, let's begin by creating the tabs and attaching all the event handlers that will be used later in the code.

Tip

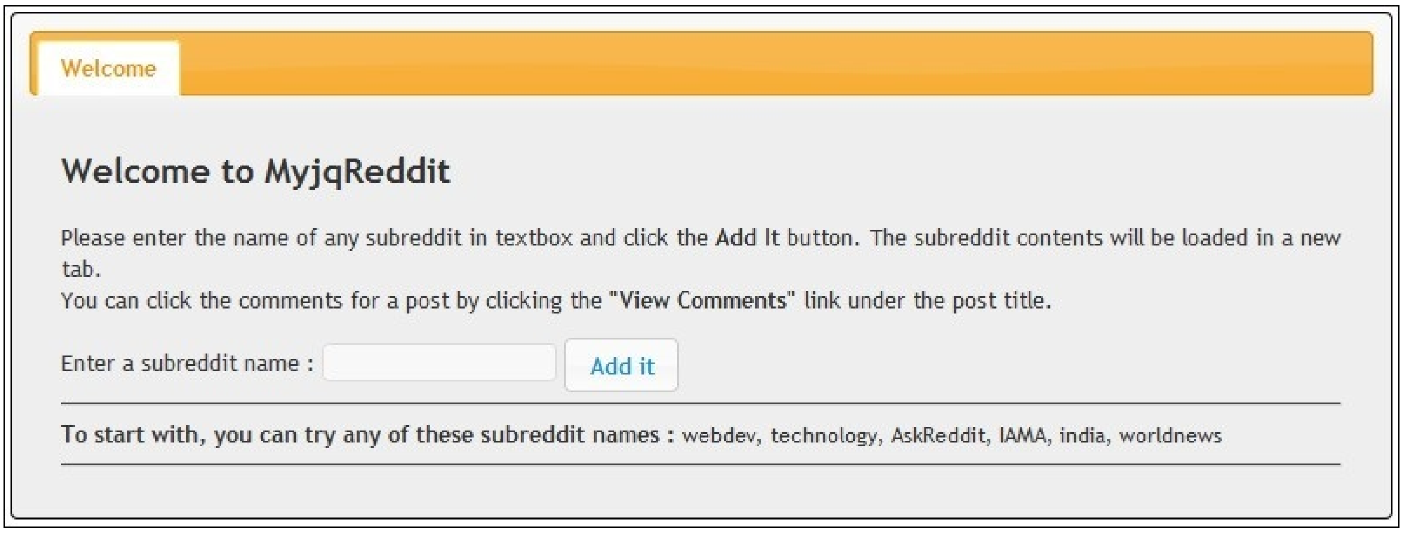
It is good practice to define all event handlers in one place. Different event handlers scattered all over the code are bad for readability, as well as for debugging the code later.

Look for the `init` method in the `myReddit.js` file and write the following code to set up tabs and event handlers:

```
$( "#redditTabs" ).tabs();

$('#addNewSubreddit').button();

$('#tabList').on('click', '.ui-icon-close', function()
{
  var tabPanelContainer = $(this).prev('a').attr('href');
  $(this).parent('li').remove();
  $(tabPanelContainer).remove();
  $( "#redditTabs" ).tabs('refresh');
});


$('#redditTabs').on('click', '#addNewSubreddit', function()
{
  myReddit.getJSONFromAPI('posts', $('#subredditName').val());
});


$('#redditTabs').on('click', '.viewText', function()
{
  $(this).parent('div').next('div.postDescription').toggle();
});

$( "#dialog" ).dialog(
{
  autoOpen: false,
  modal : true,
  title : 'Comments',
  position: { my: "center", at: "top", of: window },
  width: 800,
  height: 600
});

$('#redditTabs').on('click', '.viewComments', function()
{
  myReddit.getJSONFromAPI('comments', $(this).data('commentsid'));
  $( "#dialog" ).dialog('open');
});
```

```
$( "#dialog" ).on( "dialogopen", function( event, ui )
{
  $(this).text('Loading Comments… Please wait…');
});
```

In the first line, we created jQuery UI tabs by calling the `tabs` method on the `div` `redditTabs`. Doing so will make the tabs functional, and jQuery UI will also add various CSS classes from the CSS theme. The next line creates a jQuery UI button that we placed in the first tab. jQuery UI CSS classes will be applied to the button to make its appearance consistent with the theme. Save the file now and reload the `index.html` page in the browser. You will see that the jQuery UI theme has been applied to the tabs and the button.



Welcome to MyjqReddit

Please enter the name of any subreddit in textbox and click the **Add It** button. The subreddit contents will be loaded in a new tab.
You can click the comments for a post by clicking the **"View Comments"** link under the post title.

Enter a subreddit name :

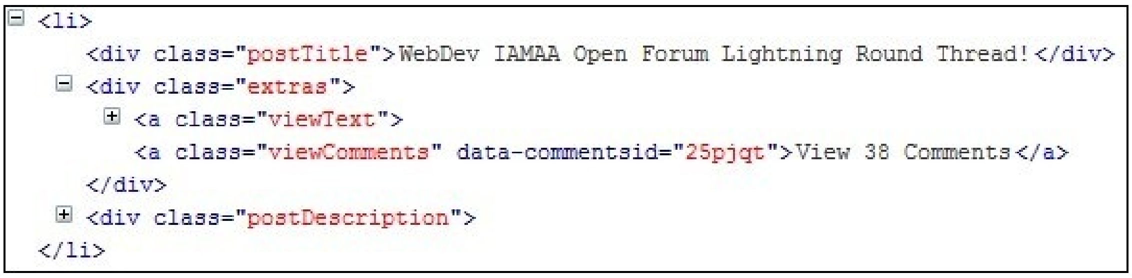
To start with, you can try any of these subreddit names : webdev, technology, AskReddit, IAMA, india, worldnews

Once a new tab is created, we will put a close icon left to the tab name to close the tab. The icon will have the base class `ui-icon` for icons and CSS class `ui-icon-close`. Hence, we have now added an event handler for the selector `ui-icon-close`.

By clicking on the close icon, we remove the tab as well as its corresponding tab panel. The first line inside the event handler finds out the corresponding tab panel for this tab by getting the value of the `href` attribute of the tab header. In the second line, we remove the parent `li` of the icon. Since `li` is what makes the tab header, removing `li` will remove the tab. Finally, we remove the tab container as well. The last line calls the `refresh` option. Remember that calling the `refresh` method is very important. This is because we have just removed a tab and we should let jQuery UI know to update the tab structure.

The next event handler is a simple one. We register the click handler on the button `addNewSubreddit`. This event handler will take the subreddit name from the input textbox with `id` as `subredditName` and will call the method `getJSONFromAPI`, which will then fetch posts for the inputted subreddit using the API. Note that we are calling `getJSONFromAPI` with two arguments. The `getJSONFromAPI` method will be called both for subreddits and comments; hence, we have passed the first argument as `posts` because we want to fetch the posts of a subreddit. The second argument is the unique name of the subreddit.

Our next event handler will be for toggling the description of a particular post. When the posts are displayed, the HTML structure of a single post will be like this:



```
⊟ <li>
      <div class="postTitle">WebDev IAMAA Open Forum Lightning Round Thread!</div>
    ⊟ <div class="extras">
        ⊞ <a class="viewText">
          <a class="viewComments" data-commentsid="25pjqt">View 38 Comments</a>
      </div>
    ⊞ <div class="postDescription">
  </li>
```

When the anchor with class `viewText` is clicked, we need to toggle the `div` with the `class postDescription`. This `div` will have a description inside it. Therefore, we are accessing its parent, which is `div` with the `class extras` and then toggling the div next to it.

With posts done, we can now move to attach event handlers for comments. As stated earlier, we are going to display comments in a dialog. Hence, we need to define a dialog and corresponding event handlers.

The `div` with `id dialog` will act as jQuery UI dialog. We have initialized it using some standard options such as `height`, `width`, `title`, and `position`. Two options need specific attention: the first one is `autopen`, which we have set to `false` because we do not want the modal to open as soon as the page loads, and the second one is the `modal` option that is set to `true`. This is to ensure that the user's focus remains solely on the dialog and they do not interact with the page while the dialog is open.

After the posts are displayed, we will need to retrieve the comments for a specific post. All the posts will have a **View XXX Comments** link underneath them, where XXX is the number of comments for the post. Each of these links will have a CSS class `viewComments` and a data attribute called `data-commentsid`. We have attached an event handler on the `click` event of the `viewComments` selector.

Clicking on this selector will call the `getJSONFromAPI` method once again. However, this time the arguments will be different. The first argument will be `comments` and the second one will be the `id` of the comment thread. After this method is called, we open the jQuery UI dialog using the `open` option.

Our last event handler is called when the dialog opens. Inside this, we simply insert some text notifying the user that comments are being loaded.

With all the event handlers in place, half of our work is done. Now, let's implement the methods that we have called in callbacks of event handlers.

Displaying posts of a subreddit

Before proceeding further, let's discuss the reddit API. Reddit provides a REST-based API and multiple response formats for the data. Out of these, the simplest is adding `.json` to a URL and accessing it. Adding `.json` at the end of any URL will get us the JSON object for that page. You can try it yourself. If you want to get the JSON for `webdev` subreddit, enter the URL http://reddit.com/r/webdev.json in your browser and you will see the JSON response. If you want to see the formatted JSON, use Google Chrome. Similarly, to get the comments JSON for any particular post, the URL becomes `http://reddit.com/comments/XYZ.json`, where XYZ is the `id` for that post. You will learn how to find the `id` as well in this section.

There is a restriction in browsers that you need to be aware of. Browsers do not allow you to make cross-domain AJAX requests. It means if your application is running on the domain `abc.com`, you cannot make AJAX requests to any other domain except that one. However, you can load images, style sheets, and other resources from any domain. Since scripts can also be loaded from any domain, they can be used as a workaround for getting responses from other domains. Suppose we created a script tag with its `src` set to http://reddit.com/r/webdev.json. However, it will simply load the JSON object inside the scrip tags, which will be useless for us. Enter JSONP, with the URL a callback function name is sent to the target domain. On receiving the callback name, the server wraps the data inside this callback name and sends it back, where it is loaded inside script tags.

For example, reddit expects the JSONP callback name to be *jsonp*. So if we set the `src` tag of script tag to http://reddit.com/r/webdev.json?jsonp=jsonp, it will return us the same JSON response in a slightly different format. It will look something like this:

```
jsonp({"kind": "Listing", "data": {} })
```

Looks familiar now? Yes, it is a function call. This means that you must have a `jsonp` function present on your page which will be passed the received data as an argument. Inside this function, you can do anything with this data now. This is how cross-domain requests are performed.

jQuery simplifies this task by providing `jsonp` as a data type and taking care of loading the data. If you have provided a callback function name in the JSONP request and have defined that function in your code, it will get executed before the success callback of the AJAX request. However, we will use the success callback in order to maintain consistency.

Let's come back to our application; once user enters a subreddit name in the textbox and click on the **Add it** button (having `id` value `addNewSubreddit`), the event handler that we defined inside the `init` method will be called. Suppose the user enters a subreddit name `webdev` in the textbox, the event handler will call the method `getJSONFromAPI` with two arguments. The first argument will be the string `posts` and the second one will be the name of subreddit, which is `webdev` in this case. Add the following code inside the `getJSONFromAPI` method:

```
var apiURL = this.apiURL;
if(type == 'posts')
{
  apiURL+= '/r/' + id + '.json';
}
else if(type == 'comments')
{
  apiURL+= '/comments/' + id + '.json';
}
else
{
  alert('Error');
  return;
}

$('#errorMessage').empty();
$('#errorContainer').hide();
$.ajax(
{
  url: apiURL,
  dataType: "jsonp",
  jsonp: 'jsonp',
  success: function(data)
  {
    if(type == 'posts')
    {
      myReddit.createTab(id, data);
    }
    else if(type == 'comments')
    {
      myReddit.displayComments(data);
    }

  },
  error: function (xhr,statusString, errorString)
  {
    $('#errorMessage').html('An error occured and content could not be
loaded.');
    $('#errorContainer').show();
  }
});
```

We stored the base `apiURL` in a local variable `url`. Next, we created the full URL to access the API. Since we are using this function for both posts and comments, we placed an `if-else` block to create the URL. This time the `if` block will get executed, as the value for the parameter type is `posts`.

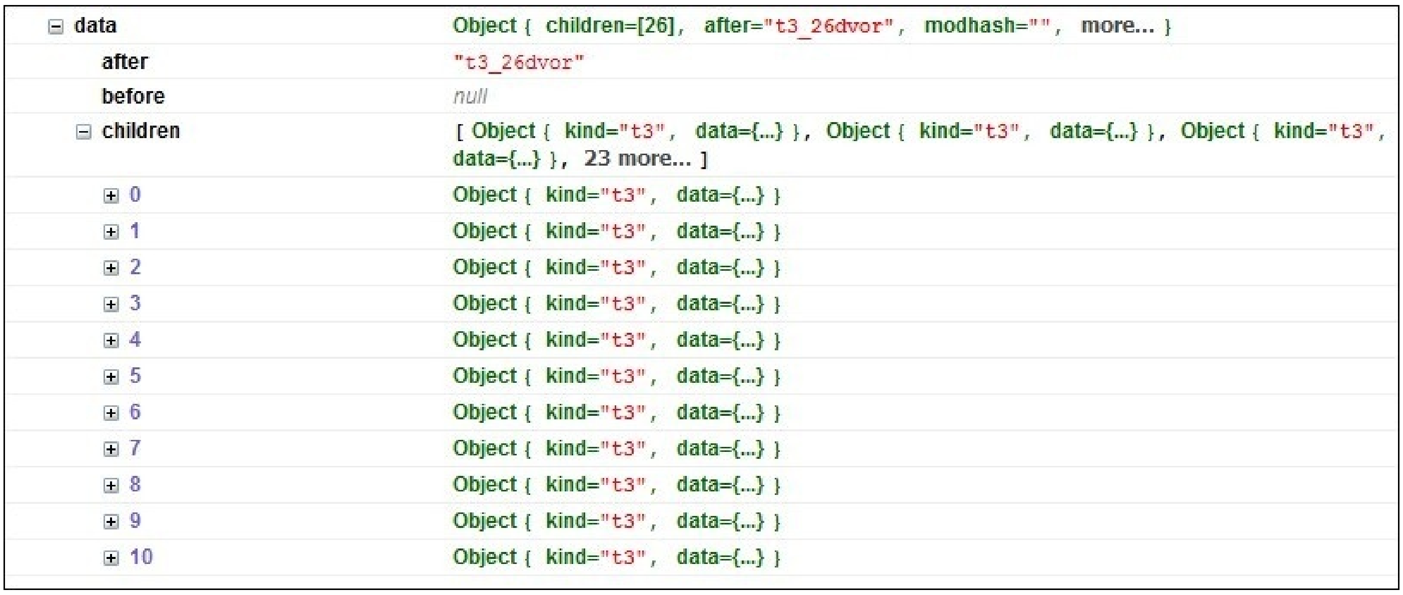
After this, we removed any error message that might be on the page and hid the `div` with `id errorContainer`.

Now, let's discuss the AJAX request that will get the data from the reddit API. We provided the URL in the variable `apiURL`. The `dataType` is `jsonp` which, as explained previously, will get the JSON response wrapped in a parameter. This parameter has also been defined in the next line using the option `jsonp`. Since reddit's API expects the callback function name in the JSONP request to be `jsonp`, we provided this value. Next we have the success and error callbacks that will be fired depending on whether the API returns a successful response or any error is encountered. Inside the `success` callback function, there is another `if-else` block. Since this request was made for posts, the `if` block will be executed and the `createTab` function will be called with the parameters `id` and `data`. Here, `id` refers to the respective subreddit names and `data` is the JSON response received from the API as a parameter. Had the request been for comments, the `else` block would have executed and `displayComments` would be called with response data as a parameter.

Inside the `error` callback, we set an error message inside the `div errorMessage` and displayed the `div errorContainer`.

Assuming a successful response has been received from API, we need to define two more methods to be able to display the posts. These functions are `createTab` and `getPostListingHtml`. The method `getPostListingHtml` will be called from inside the `createTab` method.

Before defining these functions, you need to be familiar with the response JSON structure. The response for posts of a particular subreddit is shown in the following screenshot:



```
⊟ data          Object { children=[26], after="t3_26dvor", modhash="", more... }
    after       "t3_26dvor"
    before      null
  ⊟ children    [ Object { kind="t3", data={...} }, Object { kind="t3", data={...} }, Object { kind="t3",
                data={...} }, 23 more... ]
    ⊞ 0         Object { kind="t3", data={...} }
    ⊞ 1         Object { kind="t3", data={...} }
    ⊞ 2         Object { kind="t3", data={...} }
    ⊞ 3         Object { kind="t3", data={...} }
    ⊞ 4         Object { kind="t3", data={...} }
    ⊞ 5         Object { kind="t3", data={...} }
    ⊞ 6         Object { kind="t3", data={...} }
    ⊞ 7         Object { kind="t3", data={...} }
    ⊞ 8         Object { kind="t3", data={...} }
    ⊞ 9         Object { kind="t3", data={...} }
    ⊞ 10        Object { kind="t3", data={...} }
```

As you can see, the data has the properties `after`, `before`, and `children`. We are only interested in `children` in this chapter. The `children` property is an array with each element of the array representing a post. We will see the structure of a children node when we create the DOM for posts.

Creating the tab structure

With the JSON structure clear, let’s write the code for the `createTab` method:

```
if(postList.data == null || postList.data.children == null)
{
  $('#errorMessage').html('Oops some thing is wrong');
  $('#errorContainer').show();
  return;
}

var tabContent = myReddit.getPostListingHtml(postList.data.children);
(myReddit.tabCount)++;
$('#tabList').append('<li><a href="#tabs-'+(myReddit.tabCount)+'">' +
subredditName + '</a> <span class="ui-icon ui-icon-close"
role="presentation">Remove Tab</span></li>');
$( "#redditTabs" ).append('<div id="tabs-'+(myReddit.tabCount)+'">' +
tabContent + '</div>');

$( "#redditTabs" ).tabs('refresh');

var lastTabIndex = $('#tabList li').length - 1;
$( "#redditTabs" ).tabs('option', 'active', lastTabIndex);
```

In the `createTab` method, we are receiving the data in the parameter `postList`. The first `if` block is pretty straightforward. It is checking for children in the received response. If there are no children, an error message will be displayed and we will return from the function.

The next line calls the `getPostListingHtml` method, which will be explained in the next section. We are passing the children array as the argument here. This method will create the DOM and return the HTML that will be stored in the variable `tabContent`.

Now, we need to create a tab and insert it inside the existing tab list. While defining the `myReddit` object, we created the `tabCount` property that we initially set to `1`. Since a new tab is being created now, we increment its value by one. This value is used in the next line where we create a list item with an anchor inside it. We also set the anchor’s `href` attribute and placed the subreddit name using the variable `subredditName` inside it. A `span` element that has the CSS classes `ui-icon` and `ui-icon-close` is also created. These classes are from the jQuery UI theme that will display the close icon on the right-hand side of tab. Finally, the jQuery `append` method is used to append this newly created tab to the unordered list with `id` `tabList`, where `tabList` is the list that keeps all the tab headers inside it.

After tab header, we need the corresponding tab body panel as well, which we referred to as the `href` attribute of the anchor. For this, a new `div` is created and `tabContent`, which holds the HTML for posts list, is inserted inside it. An `id` is also provided to this `div`. The value for the `id` must be the same as the value of the `href` attribute defined earlier. Finally, this `div` is appended to the `div` with `id` `redditTabs`.

Once the tab header and tab body are present in the DOM, we need to let jQuery UI know that the DOM has changed and it needs to update itself. This is done by calling the `refresh` option in the next line.

Still, the page will show the first tab as active. We will have to switch to the newly created tab. Since the newly created tabs are being appended to the end of the list, we need to find the zero-based index of the last list item. This was done by finding the number of `li` elements inside the `ul tabList` and subtracting 1 from it. So, if there are four tabs, the length will be 4 and the index of last tab will be 3, stored in variable `lastTabIndex`.

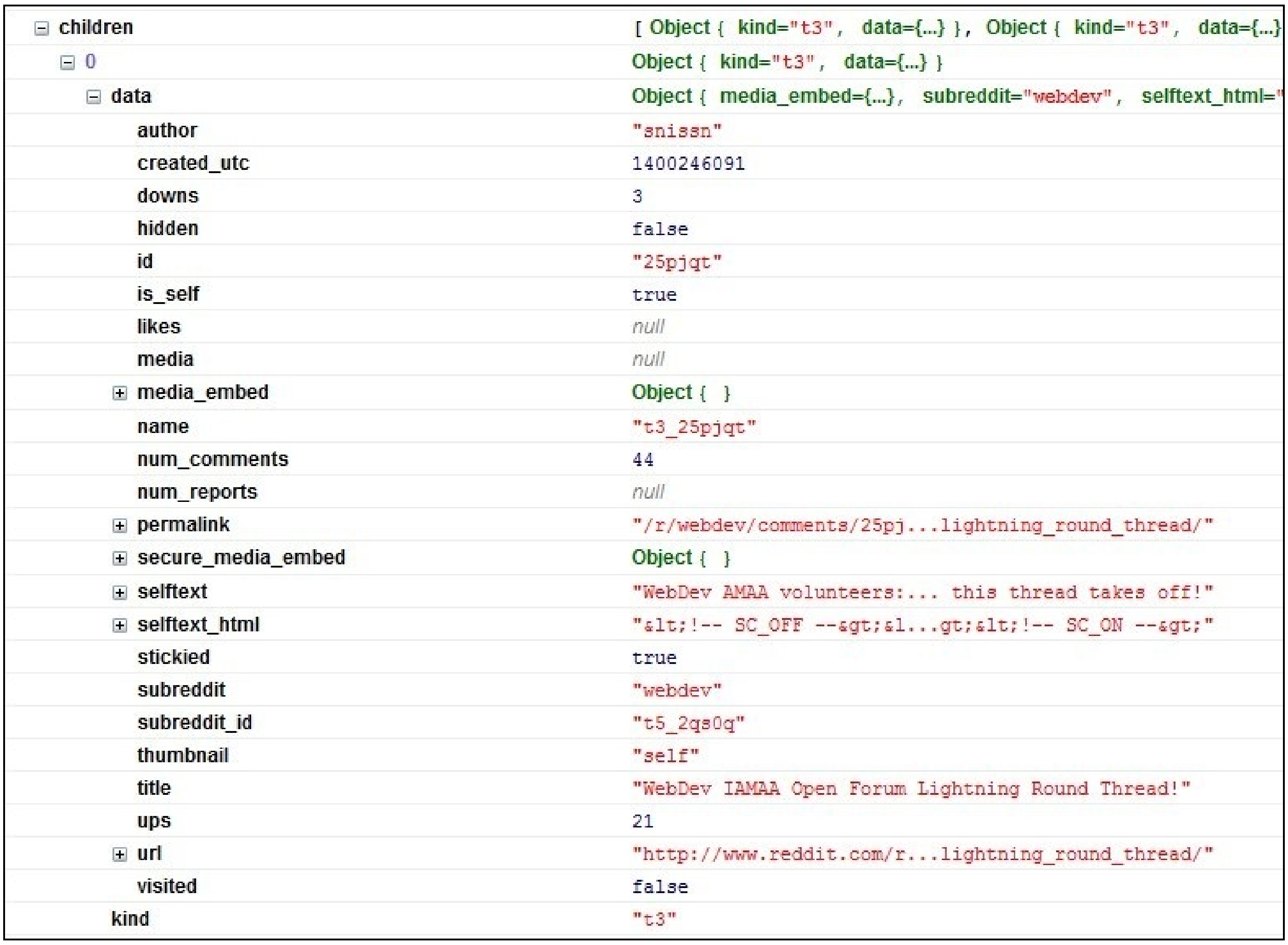
Finally, we use the active option of jQuery UI tabs to set the active tab by proving the value as `lastTabIndex`.

Before seeing any result in the browser, we need to define the `displayComments`. Buckle up once again. Only one more function to go and you will see the results.

Building the DOM for posts

In the previous section, we implemented the `createTab` method that called `getPostListingHtml` to get the HTML that displays the posts. This function receives an array of children elements from the API response as a parameter. We will iterate over these children elements and create the DOM. An unordered list will be created with one post as a list item. We will also display the description text, if available, and a link to view the comments.

We need to know beforehand the structure of a children node in order to extract data and create the HTML. The following screenshot shows the structure of a single children node:



⊟ children	[Object { kind="t3", data={...} }, Object { kind="t3", data={...}
⊟ 0	Object { kind="t3", data={...} }
⊟ data	Object { media_embed={...}, subreddit="webdev", selftext_html="
author	"snissn"
created_utc	1400246091
downs	3
hidden	false
id	"25pjqt"
is_self	true
likes	*null*
media	*null*
⊞ media_embed	Object { }
name	"t3_25pjqt"
num_comments	44
num_reports	*null*
⊞ permalink	"/r/webdev/comments/25pj...lightning_round_thread/"
⊞ secure_media_embed	Object { }
⊞ selftext	"WebDev AMAA volunteers:... this thread takes off!"
⊞ selftext_html	"&lt;!-- SC_OFF --&gt;&l...gt;&lt;!-- SC_ON --&gt;"
stickied	true
subreddit	"webdev"
subreddit_id	"t5_2qs0q"
thumbnail	"self"
title	"WebDev IAMAA Open Forum Lightning Round Thread!"
ups	21
⊞ url	"http://www.reddit.com/r...lightning_round_thread/"
visited	false
kind	"t3"

There are many more properties of a children element, but we have displayed only the relevant ones. To display the posts, we will refer to the structure displayed in the preceding screenshot.

Start by writing the following code inside the `getPostListingHtml` method to create the HTML structure:

```
var strHtml = '<ul class="postList">';
for(var i = 0; i < postListing.length; i++)
{
  var aPost = postListing[i].data;
  strHtml+= '<li>';
  if(aPost.is_self)
  {
    strHtml+= '<div class="postTitle">' + aPost.title + '</div>';
  }
  else
  {
    strHtml+= '<div class="postTitle"><a href="'+aPost.url+'"
target="_blank">' + aPost.title + '</a></div>';
  }
  strHtml+= '<div class="extras">';
  if(aPost.is_self && aPost.selftext_html != null)
  {
    strHtml+= '<a class="viewText"><span class="ui-icon ui-icon-plusthick">
</span> View Text</a>';
  }
  if(parseInt(aPost.num_comments, 10) > 0)
  {
    strHtml+= '<a class="viewComments" data-commentsid=' + aPost.id +
'>View ' + aPost.num_comments + ' Comments</a>';
  }
  else
  {
    strHtml+= '<a>No comments so far.</a>';
  }
  strHtml+= '</div>';
  if(aPost.is_self && aPost.selftext_html != null)
  {
    strHtml+= '<div class="postDescription">' +
this.htmlDecode(aPost.selftext_html) + '</div>';
  }
  strHtml+= '</li>';
}
strHtml+= '</ul>';
return strHtml;
```

Let’s go through the preceding code step by step:

- We started by declaring a variable `strHtml` and assigning the opening tag for a `ul` element. The CSS class `postList` has also been assigned to it.
- Then, there is a `for` loop to iterate over the children array. Note that there is a `data` property on the top level for each `children` node, inside which there are key-value pairs. We have taken these values in a variable `aPost`.
- We then append an opening li tag to `strHtml`.
- After this, there is an `if-else` block. Here is a property named `is_self` that tells us whether the post is a text post or a link. In reddit, a self-post means some text as title and some text as description submitted by a user. A link post means the submitted content is some text as title and a link. Therefore, if `is_self` is `true`, we append the post title in a `div` that is available in the title property of a child element. Otherwise, we create an anchor and set its `href` to `aPost.url`. Then, `aPost.title` is appended as the display text.
- Now, we want to display the text description for self posts and a link to view

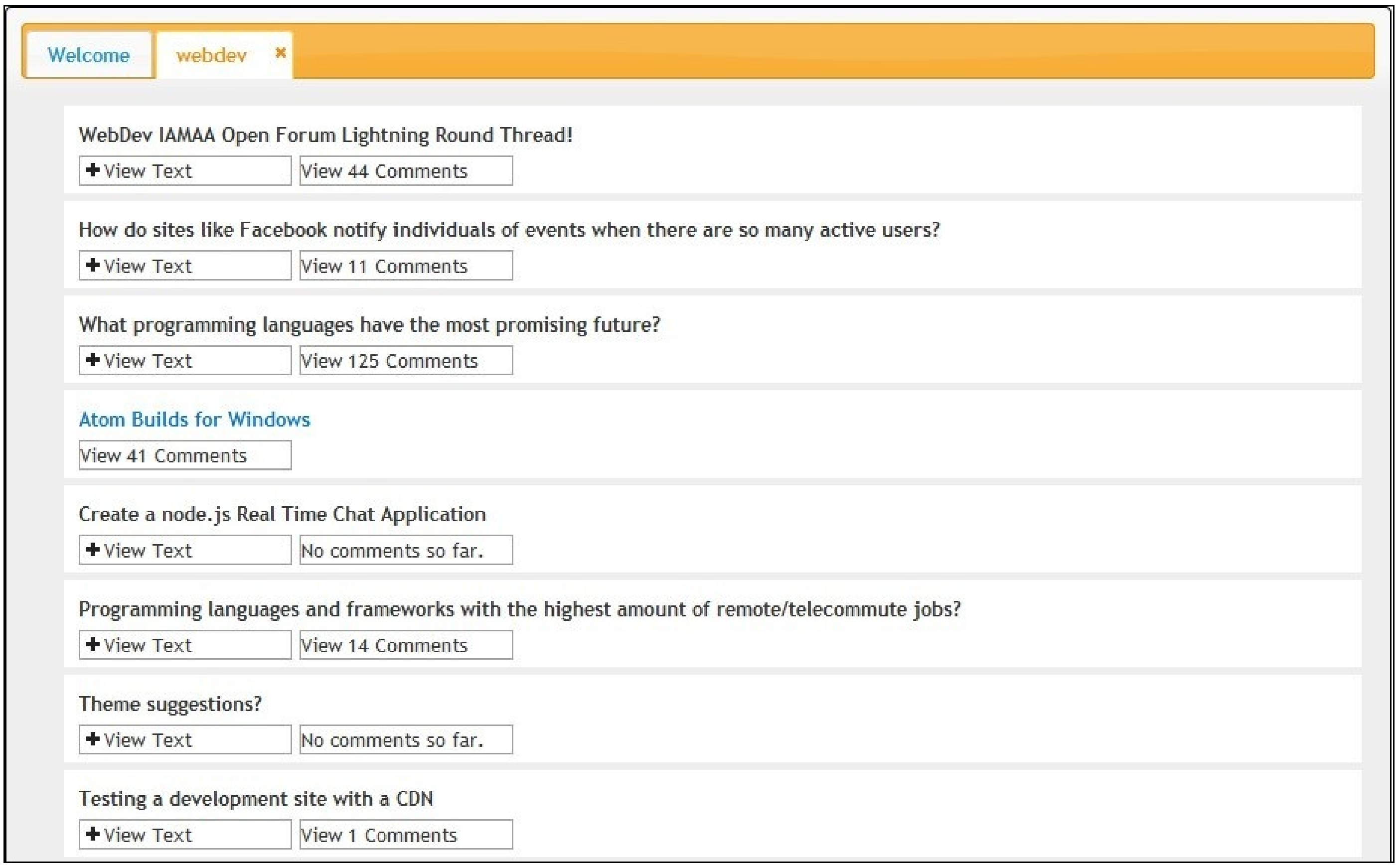
comments. For self-posts, the property `selftext_html` contains the post description. We create another `div` with the CSS `class extras`. If it is a self-post and `selftext_html` is not `null`, we create an anchor with the `class viewText`. Inside it, we place a plus icon and **View Text** as the display text. If you remember, in the `init` function, we have already defined an event handler for the selector `viewText`; hence, this class was assigned.
- There is an `if-else` block to check if there are any comments on this post. A number of comments can be determined from the value of the `num_comments` property of a child node. If comments are available, we create an anchor with the CSS `class viewComments` and a data attribute `data-commentsid`. The `data-commentsid` attribute is assigned the `id` property from child node. This is the attribute that will be used to fetch comments for this post. Therefore, make sure that it is assigned correctly.
- If there are no comments, we simply append the text **No comments so far**. The `div` with the CSS `class extras` is closed in the next line.
- Then, we display the post description if it is available. It has been kept inside a `div` with the class `postDescription`. One important point should be noted here. The `selftext_html` property contains an HTML string and this HTML is escaped. Before inserting it into the page, it will have to be escaped. This is where our little helper function `htmlDecode` is useful. We pass the HTML string to it. It simply creates a `div` element, inserts the HTML string inside it, and retrieves the inserted HTML using jQuery's `text` method. This process returns unescaped HTML that we can use to insert in our page.

Define the `htmlDecode` method as follows:

```
return $('<div/>').html(input).text();
```

The last bit is to close the `ul` tag and return the variable `strHtml` back to `createTab` method where it will be inserted into the page.

With these two methods complete, we can see the fruits of our efforts. Reload the `index.html` page, enter `webdev` in textbox, and click on the **Add it** button. Once the response is received, a new tab will be visible and you will see the post listing with all other options. The screen will resemble the following screenshot:



WebDev IAMAA Open Forum Lightning Round Thread!
✚ View Text | View 44 Comments

How do sites like Facebook notify individuals of events when there are so many active users?
✚ View Text | View 11 Comments

What programming languages have the most promising future?
✚ View Text | View 125 Comments

Atom Builds for Windows
View 41 Comments

Create a node.js Real Time Chat Application
✚ View Text | No comments so far.

Programming languages and frameworks with the highest amount of remote/telecommute jobs?
✚ View Text | View 14 Comments

Theme suggestions?
✚ View Text | No comments so far.

Testing a development site with a CDN
✚ View Text | View 1 Comments

Note that self-posts titles are black in color and link posts have a different color. Link posts will open the linked URL in a new tab. The look and feel of the posts are due to the CSS rules we defined earlier.

A couple of important points need to be mentioned here:

- Clicking on the **View Text** link will toggle the post description if it is available. This is because we already defined a related event handler in the `init` method.
- You will also be able to remove the tab by closing the cross button present on right side of the tab header. Again, this behavior is due to the event handler defined in the `init` method at the beginning of the chapter.

We will now move on to the next section where we will display the comments for a particular post in a dialog.

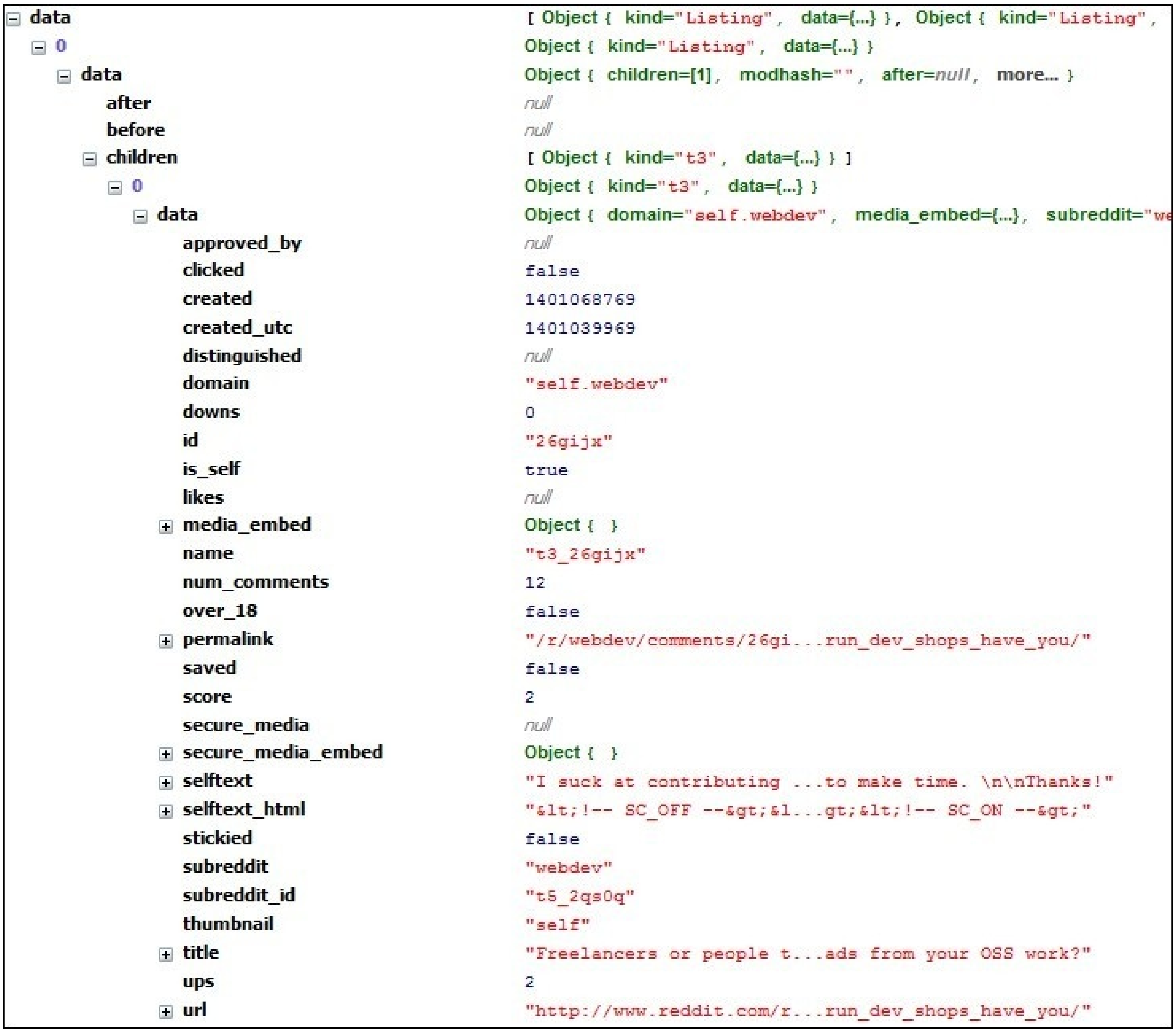
Getting comments for a post

We have already displayed the posts and a link to view comments with each post. Each of these links has a CSS `class` named `viewComments`. In the `init` method, we have defined an event handler for the `click` event on this selector. This event handler calls the method `getJSONFromAPI` with two arguments. The first argument is the set of string comments that will let `getJSONFromAPI` know that the API request has to be made for comments. The second parameter is the comments `id` for that particular post, which we assigned using the `data-commentsid` attribute. The event handler for `viewComments` gets this value and passes it as a second argument to `getJSONFromAPI`. We have already defined and discussed the working of `getJSONFromAPI`. This method will request the API for comments and will call the `displayComments` method with the API response as an argument. Since we are going to display the comments in a dialog, the event handler for the `viewComments` selector opens the dialog box as well.

The `displayComments` method will use another method called `getCommentsHTML` to create the DOM structure for comments. As mentioned in the *Getting ready* section, since comments could be nested multiple levels deep, the `getCommentsHTML` function will be called recursively to display the comments. Let's begin by defining the `displayComments` method first. Add the following code to the `displayComments` method:

```
if(data != undefined && data.length > 0)
{
  var permalink = this.apiURL + data[0].data.children[0].data.permalink;
  var linkToReddit = '<a href="'+permalink+'" target="_blank">View all the
comments on reddit</a>';
  var commentsHTML = this.getCommentsHTML(data[1].data.children);
  $('#dialog').html(linkToReddit + commentsHTML);
}
```

In the preceding code, `data` is an array containing two elements. The first element contains the metadata related to the post, such as the post's author, permalink, number of comments, and so on. The following screenshot shows its structure:



```
⊟ data                      [ Object { kind="Listing", data={...} }, Object { kind="Listing",
  ⊟ 0                       Object { kind="Listing", data={...} }
    ⊟ data                  Object { children=[1], modhash="", after=null, more... }
        after               null
        before              null
      ⊟ children            [ Object { kind="t3", data={...} } ]
        ⊟ 0                 Object { kind="t3", data={...} }
          ⊟ data            Object { domain="self.webdev", media_embed={...}, subreddit="we
              approved_by   null
              clicked       false
              created       1401068769
              created_utc   1401039969
              distinguished null
              domain        "self.webdev"
              downs         0
              id            "26gijx"
              is_self       true
              likes         null
            ⊞ media_embed   Object { }
              name          "t3_26gijx"
              num_comments  12
              over_18       false
            ⊞ permalink     "/r/webdev/comments/26gi...run_dev_shops_have_you/"
              saved         false
              score         2
              secure_media  null
            ⊞ secure_media_embed  Object { }
            ⊞ selftext      "I suck at contributing ...to make time. \n\nThanks!"
            ⊞ selftext_html "&lt;!-- SC_OFF --&gt;&l...gt;&lt;!-- SC_ON --&gt;"
              stickied      false
              subreddit     "webdev"
              subreddit_id  "t5_2qs0q"
              thumbnail     "self"
            ⊞ title         "Freelancers or people t...ads from your OSS work?"
              ups           2
            ⊞ url           "http://www.reddit.com/r...run_dev_shops_have_you/"
```

After making sure we have comments in the first place, we got the `permalink` property from metadata. Permalink is the web URL for the comments page of the related post. We added the base URL before it to make a full URL. In the next line, we created an anchor to open this comments page URL.

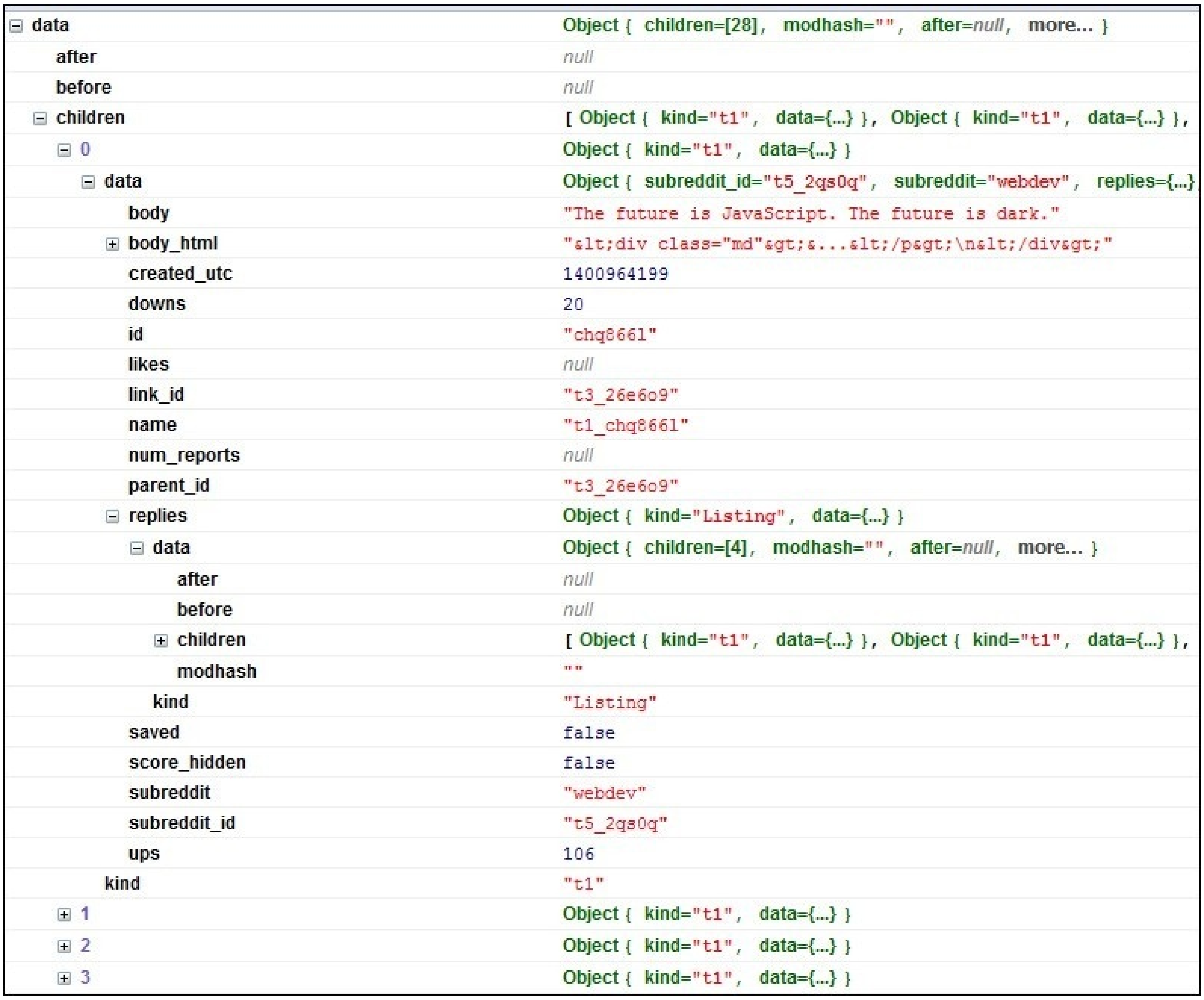
Note

Since there can be hundreds of comments for a post, reddit API sends a fixed number of comments in response. There is an important property that can be used to retrieve more comments: `after`. If you want to load more comments using a second AJAX request, you can send the value of after property as the comment `id`.

The next line calls the method `getCommentsHTML` and it will receive an HTML string in response from the method, which we assigned to `commentsHTML` variable. The last line of our method simply inserts the permalink and HTML structure for the comments into the dialog.

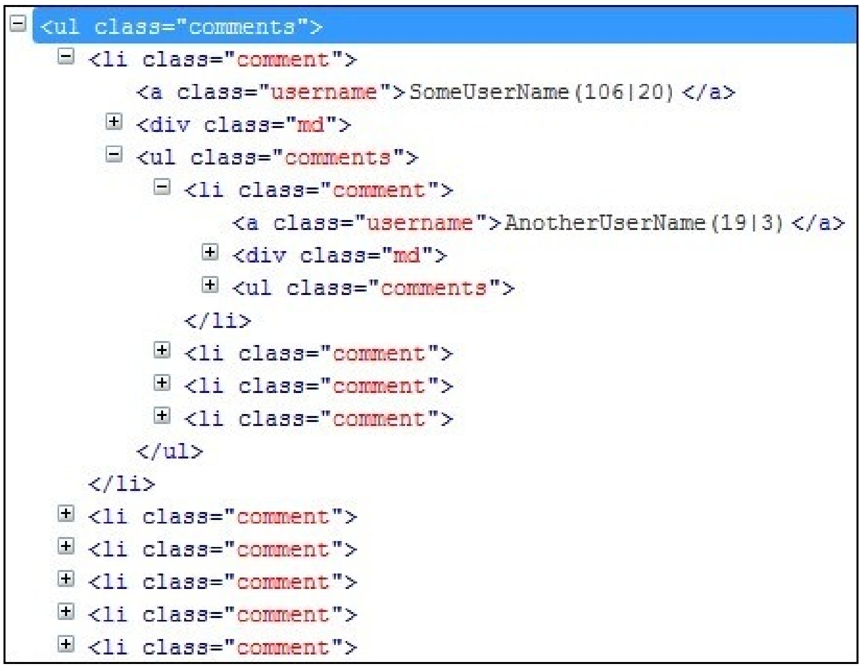
The `getCommentsHTML` method was called by passing `data[1].data.children` as an argument. The `data[1]` element refers to the second element of the API response. This element holds the actual comments inside a node that is named `children`. Each element of the `children` array is a top-level comment. To get the replies for a comment, each element of the `children` node has a property called `replies`. Again, its structure is the same as the top-level `children` element. Therefore, we can create a recursive function which will get us all the replies for a comment.

The following screenshot displays the data structure for your reference. We will refer to it while creating the DOM. Note that there are many more properties inside the data element, but we have removed those to show us only the ones relevant to us.



```
⊟ data                          Object { children=[28], modhash="", after=null, more... }
    after                       null
    before                      null
  ⊟ children                    [ Object { kind="t1", data={...} }, Object { kind="t1", data={...} },
    ⊟ 0                         Object { kind="t1", data={...} }
      ⊟ data                    Object { subreddit_id="t5_2qs0q", subreddit="webdev", replies={...}
          body                  "The future is JavaScript. The future is dark."
        ⊞ body_html             "&lt;div class="md"&gt;&...&lt;/p&gt;\n&lt;/div&gt;"
          created_utc           1400964199
          downs                 20
          id                    "chq8661"
          likes                 null
          link_id               "t3_26e6o9"
          name                  "t1_chq8661"
          num_reports           null
          parent_id             "t3_26e6o9"
        ⊟ replies               Object { kind="Listing", data={...} }
          ⊟ data                Object { children=[4], modhash="", after=null, more... }
              after             null
              before            null
            ⊞ children          [ Object { kind="t1", data={...} }, Object { kind="t1", data={...} },
              modhash           ""
            kind                "Listing"
          saved                 false
          score_hidden          false
          subreddit             "webdev"
          subreddit_id          "t5_2qs0q"
          ups                   106
        kind                    "t1"
    ⊞ 1                         Object { kind="t1", data={...} }
    ⊞ 2                         Object { kind="t1", data={...} }
    ⊞ 3                         Object { kind="t1", data={...} }
```

The HTML structure for comments will be an unordered list `ul` with each top-level comment being a list item `li`. Inside this `li`, we will display the username along with up and down votes for the comment. The actual comment made by the user will be next to it. If there are any replies for this comment, another `ul` will be created following the same HTML structure. The HTML structure we are targeting to achieve is shown in the following screenshot:



```
<ul class="comments">
  <li class="comment">
      <a class="username">SomeUserName(106|20)</a>
    <div class="md">
    <ul class="comments">
      <li class="comment">
          <a class="username">AnotherUserName(19|3)</a>
        <div class="md">
        <ul class="comments">
      </li>
      <li class="comment">
      <li class="comment">
      <li class="comment">
    </ul>
  </li>
  <li class="comment">
  <li class="comment">
  <li class="comment">
  <li class="comment">
  <li class="comment">
```

Now, let's define the `getCommentsHTML` method to create this HTML. The following code will be used to create the HTML structure for comments:

```
getCommentsHTML : function(commentsList)
{
  var str  = '<ul class="comments">';
  for(var i = 0; i< commentsList.length; i++)
  {
    var x = commentsList[i];
    str+= '<li class="comment"> <a class="username">' + x.data.author +
'('+x.data.ups+'|'+x.data.downs+')</a> ' +
this.htmlDecode(x.data.body_html);
    if(x.data.replies != undefined && x.data.replies != "")
    {
      str+= this.getCommentsHTML(x.data.replies.data.children);
    }
    str+= '</li>';
  }
  str+= '</ul>';
  return str;
}
```

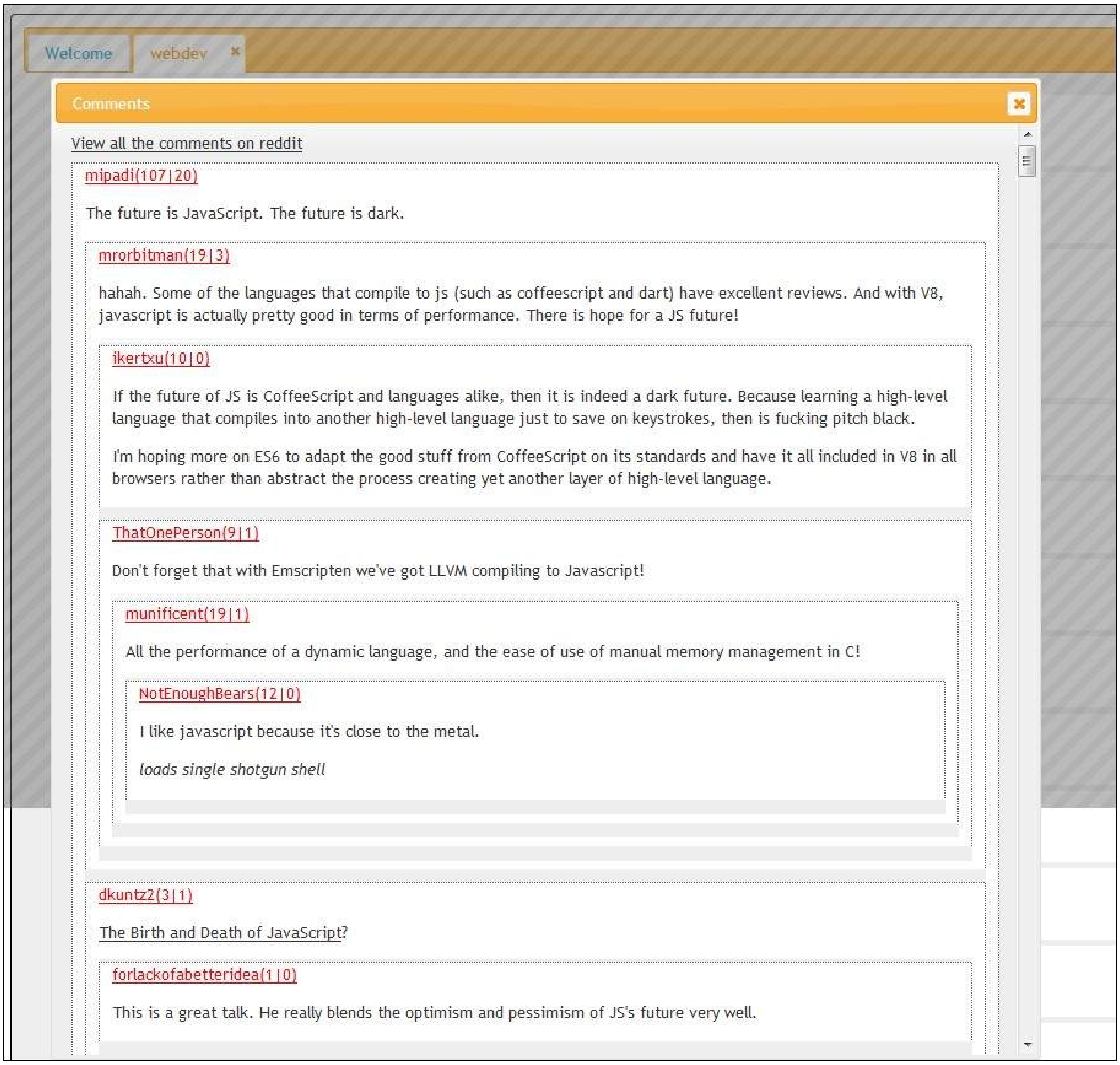
We receive a `commentsList` parameter that contains the top-level `children` array. To create HTML, we begin by creating a string named `str` that will keep storing the HTML strings as we create it. A top-level `ul` is created and the CSS class `comments` is assigned to it. Then, we iterate over the elements in the `commentsList` array, create a `li` element for each list item, and assign CSS class `comment` to it. Inside this `li`, an anchor with class name `username` is created, where we place the username and up and down votes for the comments that are available in the `author`, `ups`, and `downs` properties, respectively. The comment body is available in the `body_html` property that we have unescaped using the helper function `htmlDecode`.

The conditional statement checks if there are any replies to this comment. If the `replies` property is not empty, we call the `getCommentsHTML` method again and pass the `children` array of the `replies` property to it. This method will keep getting called until there are no replies for any children and the resulting HTML will keep getting added to the variable `str`. The `li` element is closed in the last line of the `for` loop.

After the loop is finished, we close the `ul` element as well and return the HTML created so far back to the `displayComments` method, where it is inserted into the DOM.

Our minimal reddit clone **MyjqReddit** is complete now and you can check the comments for a post as well. Reload the `index.html` page in your browser and enter a subreddit name (for example, `webdev` or `technology`) in the textbox and click on the **Add It** button. From the post listing in the new tab, click on the **View XXX Comments** link. A dialog will appear with the text **Loading Comments… Please wait…** and after the response is received, you will see the comments in dialog. Due to the CSS classes we have applied to the `ul`, `li`, and other elements, you will see that child comments are shifted to the right as the comment level gets deeper.

Here is a screenshot of what you will see in the case of large number of comments:



View all the comments on reddit

mipadi(107|20)

The future is JavaScript. The future is dark.

mrorbitman(19|3)

hahah. Some of the languages that compile to js (such as coffeescript and dart) have excellent reviews. And with V8, javascript is actually pretty good in terms of performance. There is hope for a JS future!

ikertxu(10|0)

If the future of JS is CoffeeScript and languages alike, then it is indeed a dark future. Because learning a high-level language that compiles into another high-level language just to save on keystrokes, then is fucking pitch black.

I'm hoping more on ES6 to adapt the good stuff from CoffeeScript on its standards and have it all included in V8 in all browsers rather than abstract the process creating yet another layer of high-level language.

ThatOnePerson(9|1)

Don't forget that with Emscripten we've got LLVM compiling to Javascript!

munificent(19|1)

All the performance of a dynamic language, and the ease of use of manual memory management in C!

NotEnoughBears(12|0)

I like javascript because it's close to the metal.

loads single shotgun shell

dkuntz2(3|1)

The Birth and Death of JavaScript?

forlackofabetteridea(1|0)

This is a great talk. He really blends the optimism and pessimism of JS's future very well.

Improving MyjqReddit

In this chapter, we deliberately left out a few things in order to focus more on jQuery UI and less on API. You can take these as an exercise to improve the mashup we created in this chapter. Here are a few suggestions to take it to the next level:

- We display only the first set of comments for a particular post. Use the API to load more comments by making new requests to the API using the value of `after` property.
- Modify the code so that if the linked URL of any post is an image, that image should be displayed in a modal dialog if the URL is clicked.
- Avoid opening duplicate tabs with the same subreddit name. For example, if the `webdev` subreddit is already open in a tab, disallow any fresh requests for it and switch to the corresponding tab instead.
- Use the `progressbar` component to display an animated progress bar when an AJAX request is being sent.

Summary

This chapter was a long read focusing on the tab and dialog components. You worked with tabs and dialog, learned to create dynamic tabs, and in this process learned about JSONP and the reddit API as well. You must be comfortable with the object literal pattern by now.

We will see another challenge in the next chapter, where we will create three different implementations of CAPTCHA.

Chapter 5. Implementing CAPTCHA using Draggable and Droppable

In the previous chapter, we created a reddit news reader that was a bit complex from a coding point of view. To compensate for this, we will reduce our pace a bit in this chapter and develop various implementations of CAPTCHA.

CAPTCHA (short for **Completely Automated Public Turing test to tell Computers and Humans Apart**) is a form of test where web forms present questions to users that are supposed to be solved by humans only.

Imagine a simple registration form of a website. If you know all the fields present in the page, you can write a script to register as many times as you can.

Now, suppose a CAPTCHA challenge is placed in the registration form, where the user is shown 10 different colors and is asked to select a random color. A regular bot will not be able to do so and hence the registration will fail.

Note

The main purpose of CAPTCHAs is to present such tests that only human users could pass. Any scripts or bots should not be able to pass the CAPTCHA test.

In this chapter, we will focus on how we can create simple CAPTCHA implementations using some of the components from the jQuery UI library.

We will create three different types of CAPTCHA implementations. In all the implementations, we will generate CAPTCHA values on the server side and store them in session. Then, the page will be displayed to the user and user will validate the CAPTCHA and submit the form. The submitted value will then be validated against the value in the session. On successful validation, a success message will be displayed but an error message is displayed if CAPTCHA fails. Here are the variations that we will implement in this chapter:

- **Draggable color CAPTCHA**: The user will be shown five different colored boxes. A color name will also be written. To validate CAPTCHA, the user will have to drag the said colored box to a droppable box. After this, the form will be submitted and the values will be validated on the server side. jQuery UI's draggable and droppable components will be used to built this.
- **Slider CAPTCHA**: Two numbers between 0 and 100 will be generated from the server side that will be kept in session and will be shown to the user as well. In the page, there will also be a slider component. The user will have to drag the slider and set its minimum and maximum values to the given values. After setting the slider values, the form will be submitted and values set on the slider will be matched against the session values. This implementation will use jQuery UI's slider component.
- **Number CAPTCHA**: We will make this CAPTCHA a bit more difficult than the

previous two versions. We will generate a five-digit number from the server-side script but this number will not be printed on the page directly. We will use a CSS sprite image to display this number using five different images. Under these images, we will display five more numbers that will actually be the images of digits of original number. The user will have to arrange these images in the order they appear in original five-digit number. After this, the form will be submitted and the number formed by arranging the images will be validated against the five-digit number that is in session. jQuery UI's sortable component will be used for this implementation.

Note

Since the CAPTCHA values need to be generated from a server-side script, we will use PHP as server-side script in these examples. Even if you are not familiar with PHP, you will not face any difficulty in following it, as the code for generating values is simple and will be explained in full detail. The majority of the work will be done in jQuery only. The logic is the same and you can use any other scripting language on the server side to achieve the same effect.

Now that we are clear about what we are going to create, let's write the code and create the CAPTCHAs one by one. We will start with the draggable color CAPTCHA.

Creating the folder structure

As we did in the previous chapters, we will first create the folder structure.

1. Create a folder named `Chapter5` inside the `MasteringjQueryUI` folder.
2. Copy the `js` and `css` folders from the downloaded jQuery UI folder inside the `Chapter5` folder.
3. Inside the `Chapter5` folder, create a PHP file named `colorCAPTCHA.php` for the first CAPTCHA.

   ### Note

   Since we are using PHP as server-side language, you will need PHP installed on your machine along with a web server, such as Apache, to run these files. You can use a software bundle, such as WAMP or XAMPP, to install all at one go.

Implementing the drag and drop CAPTCHA

For our first implementation, as mentioned previously, we will first create an array of five color names. Then, we will generate a random number between zero and four and pick the color on that index from the array. This color name will be saved in session for validation later on. Then, we will create five colored boxes in HTML and set their background color using the color names defined in the array earlier. We will then create a separate div where the user will drag and drop a colored box. Finally, we will make the dragging and dropping functional.

As far as rules go, we will apply two restrictions: the droppable box could have only one colored box inside it at a time and second, users should also be able to remove an earlier dropped colored box from the droppable container and drop another box.

Setting a random color

Let’s begin by create the colored array first. Open the file `colorCAPTCHA.php` in your editor and write the following PHP code:

```
<?php
  session_start();
  $arrColors = array('red', 'green', 'blue', 'white', 'black');
  $randomKey = array_rand($arrColors);
  $randomColor = $arrColors[$randomKey];
  $_SESSION['randomColor'] = $randomColor;
?>
```

In PHP, the `session_start` function is used to create a new session or access a session if it already exists. By using `session_start` in the first line, we created a new session. In the next line, we defined an array named `$arrColors` that contains names of five commonly known colors. After this, we picked a random key from an array using the `array_rand` function of PHP. Using this random key, we select the corresponding color name from the array, which we stored in variable `$randomColor`. This variable will be used to let the user know which colored box he/she has to drag and drop.

The last line stores the generated random color name in session. It will be used after the page is submitted to check against the user-submitted value.

Displaying the CAPTCHA

We have the `colors` array ready, and a random color name from it as well. We can now use this data to create our page. Along with the HTML markup, we will need some CSS styling to make the elements look better. Here is the code that will be used to create the HTML. Add this code after the closing tag of PHP written in previous section:

```
<html>
  <head>
    <meta charset="utf-8">
    <title>Color CAPTCHA</title>
    <link rel="stylesheet" href="css/ui-lightness/jquery-ui-
1.10.4.custom.min.css">
    <style type="text/css">
    body{
      font-family:arial,verdana;
      font-size:12px;
      margin: 0px auto;
      width: 700px;
    }
    #frmCAPTCHA{
      border: 1px solid #aaa;
      float: left;
      margin: 0 auto;
      padding: 20px;
      width: 100%;
    }
    h3{
      border-bottom:1px solid #aaa;
    }
    .row{
      display: block;
      padding: 20px 10px;
      clear:left;
      float:left;
    }
    .colors{
      float: left;
      border: 1px solid #aaa;
      padding: 20px 10px;
    }
    .colorTile{
      border:1px solid #000;
      margin:0 5px;
      display: block;
      float: left;
      height: 40px;
      width: 40px;
      z-index:1;
    }

    .row label{
      float: left;
      padding: 0px 10px;
      width: 25px;
      text-align: center;
    }

    .dropbox{
      border: 1px solid #aaa;
      float: left;
      height: 82px;
      width: 100px;
      z-index: 0;
    }

    .clear{
      clear:both;
    }
  </style>
  </head>
  <body>
    <form id="frmCAPTCHA" method="post">
      <h3>Color CAPTCHA</h3>

      <div class="row">
        <div class="colors">
          <?php
            foreach($arrColors as $color)
            {
            ?>
          <div class="colorTile" style="background-color:<?php echo
$color;?>;" data-key="<?php echo $color;?>"></div>
          <?php
            }
          ?>
        </div>
      </div>

      <div class="row">
        <div class="dropbox">Drop here </div>
        </div>
      </div>

      <div class="row">
        <strong>Solve the CAPTCHA by dragging the <strong><u><?php echo
strtoupper($randomColor);?></u></strong> colored box in the box above.
</strong>
      </div>

      <div class="row">
        <input type="hidden" name="selectedColor" id="selectedColor"/>
        <button type="submit" name="submit">Check</button>
      </div>
    </form>

    <script src="js/jquery-1.10.2.js"></script>
    <script src="js/jquery-ui-1.10.4.custom.min.js"></script>
  </body>
</html>
```

In the preceding code, inside the `head` section, we included the jQuery UI CSS file. Then, we have custom CSS styles. These CSS rules create a 700 pixels-wide document that is centered on the page. Various other classes are also defined to apply borders and fix the height and width of elements.

In the `body` section, we start by creating a form and setting its `id` as `frmCAPTCHA`. Inside this form, there is an `h3` element that acts a page heading. There is a div, inside which we will display the colored boxes. The div has the class `row` and it has another div with the class `colors` inside it. The CSS class `row` as defined in the `style` section has been applied specifically to make the div behave as rows. Inside the div with the class `colors`, we have a `foreach` loop that iterates in the array `$arrColors` and creates a div for each array element. Note that the array `$arrColors` holds the names of the five colors. Each div that is being created is being assigned a `colorTile` class and a background color. The CSS serves two purposes here. First, it defines the height, width, and a few other CSS properties that make the div look like a square box. Second, this class will be used to implement the draggable behavior later on for other purposes.

An important thing should be noted here. We are also defining a data attribute for the div called `data-key`. The value of this attribute is the name of color. We will get back to this in the following sections.

Another div with the class `row` is defined in the code, along with a div with the class `dropbox` inside it. The class `dropbox` has been defined in the `style` section to make this div a box of size 100 x 100. This div will be converted to a droppable container where users will drag the colored boxes.

In the next row, we display a message to the user to tell them which color has to be selected. Remember that we have the color name in the variable `$randomColor`.

In the last row, we have a hidden input field and a submit button. The hidden field has its `name` as well as `id` set to `selectedColor`. This hidden field will be used to hold the value of colored box that has been dropped by the user on the droppable container. The submit button has the name `submit`.

Finally, we included the main jQuery file and the jQuery UI JavaScript file.

Making the drag and drop functional

In order to comply with the rules we have laid out in the beginning of this section, we will need one draggable and two droppable components. The colored boxes have the class `colorTile` that will be made draggable. For droppable, users can drop the colored boxes into div with the class `dropbox` and they can also drag a dropped box from `dropbox` to the original list of colors. Hence, two droppables will be required. After clicking on the **Check** button, we will have to fill the hidden field with the value of color that is present in `dropbox`.

Add the following code in the file, after the line where jQuery UI's JavaScript file has been loaded:

```
<script>
  $(document).ready(function(){
    $('button').button();
    $('.colorTile').draggable({
      revert : 'invalid',
      helper: 'clone',
      cursor: 'move'
    });

    $('.dropbox').droppable({
      accept: function(item){
        if(item.hasClass('colorTile') && !$('.dropbox .colorTile').length)
        {
          return true;
        }
        return false;
      },
      activeClass: 'ui-state-highlight',
        drop: function( event, ui )
      {
        var $item = (ui.draggable);
        $item.css({'left' : '0', 'top' : 0}).appendTo('.dropbox');
      }
    });

    $('.colors').droppable({
      accept: '.colorTile',
      drop: function( event, ui )
      {
        var $item = (ui.draggable);
        $item.css({'left' : '0', 'top' : 0}).appendTo('.colors');
      }
    });

    $('#frmCAPTCHA').submit(function(){
        var x = $('.dropbox .colorTile').data('key');
        $('#selectedColor').val(x);
    });
  });
</script>
```

First of all, we converted the button into a jQuery UI button. This creates a look and feel for the button as per jQuery UI's theme.

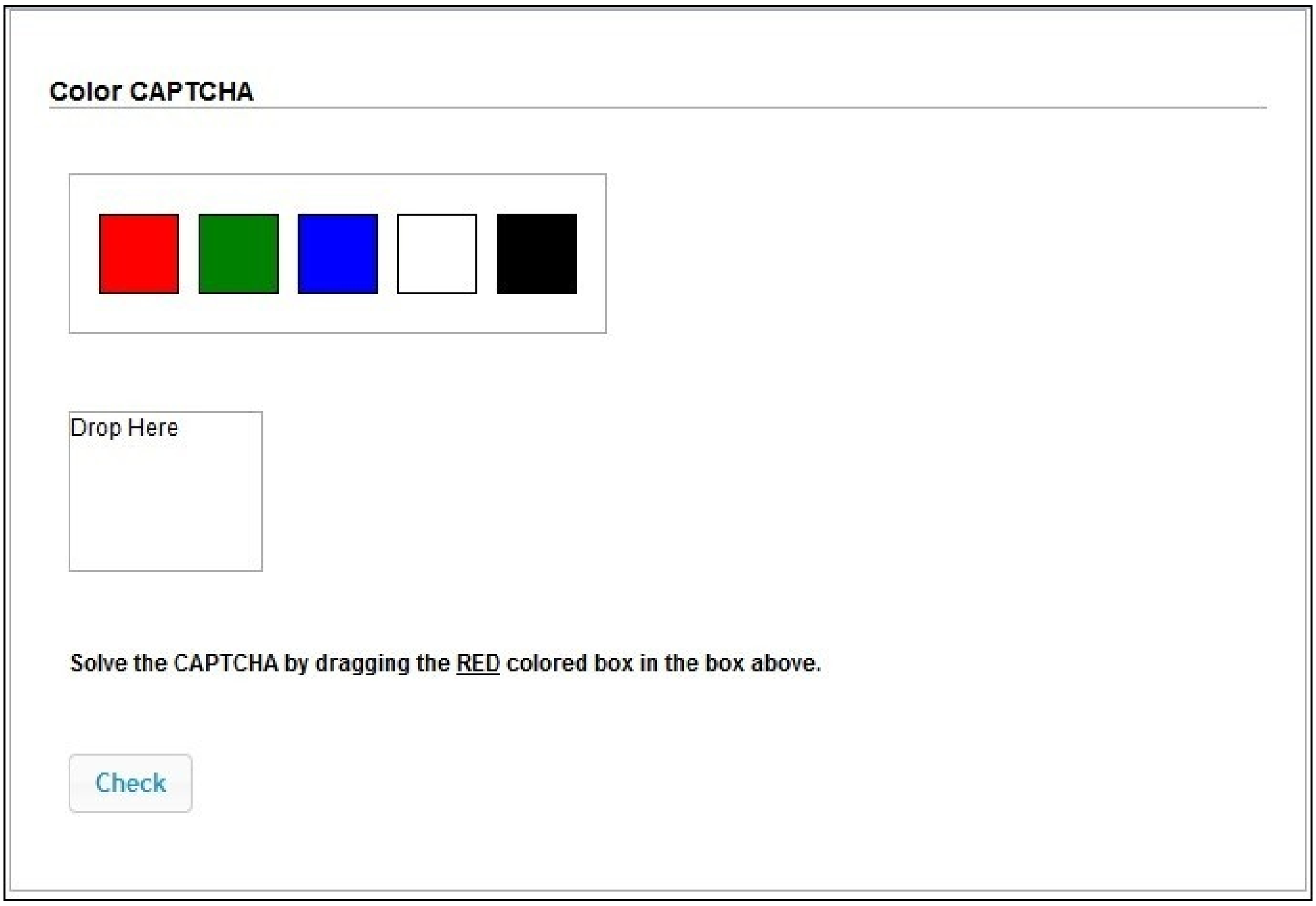
Next, we made the colored boxes with the draggable class `colorTile`. We also provided three options for the draggable component. The first option is `revert`, which has been set to `invalid`. This means that draggable will revert to its original position if a drop action is not performed. The second option is `helper`, which we have set to `clone`. Setting `helper` to `clone` creates a duplicate copy of the element being dragged and drags the clone instead of the original one. Once drop is done, the clone is placed in a new drop position and the original element is removed. This gives a good visual feel to the user, as the UI changes only after a drop is performed. The last option is `cursor`, which has been set to `move`. This will change the cursor as soon as dragging starts.

We made the div with the class `dropbox` droppable. The drop can happen only if the div being dragged has the class `colorTile` and `dropbox` has no `colorTile` divs inside it. We defined an `accept` method to achieve this. Inside the `achieve` method, the first part of the `if` condition checks whether it is indeed a div with the class `colorTile` that is being dragged. The second part checks if there are any divs with the class `colorTile` inside `dropbox`. If both conditions are met, `true` is returned, which means the drop has been accepted. The `activeClass` option has been assigned the value `ui-state-highlight`. This is a class name of the jQuery UIs CSS framework. Providing a class name to the `activeClass` options assigns the class name to droppable, while an accepted element is being dragged. Finally, there is the `drop` function. Normally, we would not require a `drop` method as the div `dropbox` is already a droppable for div with the classes `colorTile`. However, since we want to allow dragging from the div dropbox as well, we will have to detach the div `colorTile` from DOM and append it to `dropbox`. Inside the `drop` method, we are setting the `top` and `left` CSS attributes of an accepted `colorTile` to `0` and appending it to `dropbox`. Setting these values to `0` for the div `colorTile` is required. This is because while being dragged, they were positioned inside the div with the class `colors` and dragging will cause the `top` and `left` values to change. Since we are appending it inside `dropbox`, the old `left` and `top` values should not be retained.

The second droppable is the div with the class `colors` itself that originally holds all divs with the class `colorTile`. This droppable will be used when a user drags a div with `id` `colorTile` from `dropbox` to div with the class `colors`. Like the previous droppable, we have set its `accept` option to `.colorTile` and have similarly defined a drop method where we are setting its `top` and `left` values to `0` and appending it inside the div with class `colors`.

In the end a `submit` event handler for form is defined. Remember we defined a data attribute `data-key` for each draggable div. This event handler is checking inside the `dropbox` div if there is any `colorTile` present. If it is, we take its `key` value and set it as the value of the hidden input with the `id` and `name` `selectedColor`, which is present just before the submit button.

We can now check how our page looks. Save the file and access the file through your web server. You will see a screen like this:



Solve the CAPTCHA by dragging the RED colored box in the box above.

Clicking on the **Check** button will submit the form but nothing will happen, as we have not yet validated the CAPTCHA.

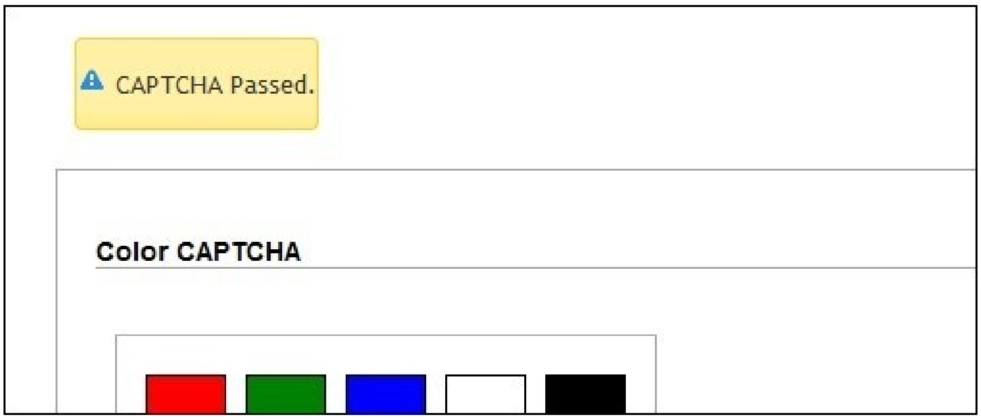
Validating on the server

To check whether the CAPTCHA has passed or failed, we need to compare the value of the color that is present in the session to the one which was set in the hidden input `selectedColor`. In the `colorCAPTCHA.php` file, write the following code just after `session_start()` method:

```
if(isset($_POST['submit']))
{
  if($_POST['selectedColor'] != $_SESSION['randomColor'])
  {
?>
  <div class="row ui-widget" style="line-height: 20px;">
    <div class="ui-state-error ui-corner-all">
      <p>
      <span class="ui-icon ui-icon-alert" style="float: left; margin-right:
.3em;"></span>
        CAPTCHA Failed. Try again.
      </p>
    </div>
    </div>

  <?php
    }
    else
    {
  ?>
  <div class="row ui-widget" style="line-height: 20px;">
    <div class="ui-state-highlight ui-corner-all">
      <p>
        <span class="ui-icon ui-icon-alert" style="float: left; margin-
right: .3em;"></span>
        CAPTCHA Passed.
      </p>
    </div>
  </div>

  <?php
  }
}
```

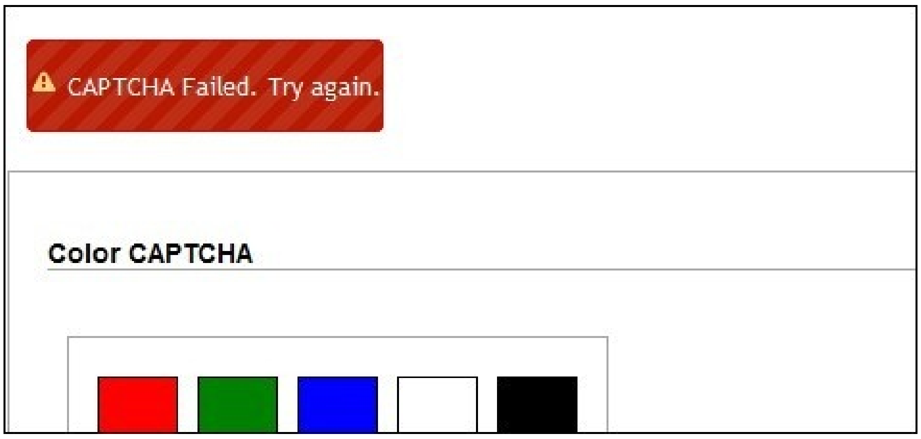
The first line of the code checks whether the form was submitted. Inside it, we take the value stored in the session (from the session variable `randomColor`) and the value posted from the form (which will be available via `$_POST['selectedColor']`) and compare them. If they match, a jQuery UI themed success message is displayed; otherwise, an error message is shown. The success and error message will appear at the top of the page.

A success message is displayed first:


CAPTCHA Passed.

Color CAPTCHA

The following error message is shown if CAPTCHA fails:


CAPTCHA Failed. Try again.

Color CAPTCHA

Improving the color CAPTCHA

The preceding CAPTCHA is a basic one and can be improved. Just try to think of new and unusual ways in which components can be used. Here are a few suggestions to get you started:

- Since not all people might recognize colors, use pictures of daily objects such as bikes, cars, cats, chocolate, and so on.
- Allow the user to drop things in groups. Like, show 3 apples, 3 oranges and 3 mangoes and ask to drop 1 apple and 2 oranges in the `dropbox`.

Creating the slider CAPTCHA

In our second implementation, we will use jQuery UI’s `slider` component to create a CAPTCHA. Two numbers between 0 and 100 will be generated from the server and will be stored in session. In the page, a range slider with two handles from 0 to 100 will be displayed. The user will have to drag the slider handles and set the slider values so that they match the values generated from the server side.

Generating minimum and maximum values for the slider

Inside the `Chapter5` folder, create a new file named `sliderCAPTCHA.php` and start by adding the following code to it:

```
<?php
  session_start();
  $randomNumber1 = (string)rand(0, 49);
  $randomNumber2 = (string)rand(50, 100);
  $_SESSION['sliderMin'] = $randomNumber1;
  $_SESSION['sliderMax'] = $randomNumber2;
?>
```

The first line after the PHP opening tag `<?php` is a call to the `session_start` function (which you will remember from the previous CAPTCHA implementation). In the next two lines, we generate two random numbers using the `rand` function of PHP. The first random number will be between 0 and 49, and the second number will be between 50 and 100. Both of these numbers are then stored in session in the session keys `sliderMin` and `sliderMax`, respectively, to be validated later.

Making the slider functional

After getting two random numbers, we can now proceed to create the slider itself. We will now create the HTML file that will display the values generated from the server, the slider, and a button to submit the form. Just below the closing tag of PHP, add the following code to create the page:

```
<html>
  <head>
    <meta charset="utf-8">
    <title>Slider CAPTCHA</title>
    <link rel="stylesheet" href="css/ui-lightness/jquery-ui-
1.10.4.custom.min.css">
    <style type="text/css">
      body{
        font-family:arial,verdana;
        font-size:12px;
        margin: 0px auto;
        width: 500px;
      }
      .frmCAPTCHA{
        border: 1px solid #EEEEEE;
        float: left;
        margin: 0 auto;
        padding: 20px;
        width: 100%;
      }
      h3{
        border-bottom:1px solid #eee;
      }
      .row{
        display: block;
        padding: 20px 10px;
      }
      .row label{
        float: left;
        padding: 0px 10px;
        width: 25px;
        text-align: center;
      }

      #slider{
        width:300px;float:left;
      }
      .clear{
        clear:both;
      }
    </style>
  </head>
  <body>
    <form class="frmCAPTCHA" method="post">
      <h3>Slider CAPTCHA
        <br>
        <small>Set the minimum and maximum values of slider to <?php echo
```

```
$randomNumber1;?> and <?php echo $randomNumber2;?> respectively.</small>
      </h3>

      <div class="row">
        <label class="minVal">0</label>
        <div id="slider"></div>
        <label class="maxVal">100</label>
      </div>

      <div class="row">
        <input type="hidden" name="minValSelected" id="minValSelected"/>
        <input type="hidden" name="maxValSelected" id="maxValSelected"/>
        <button type="submit" name="submit">Check</button>
      </div>
    </form>

    <script src="js/jquery-1.10.2.js"></script>
    <script src="js/jquery-ui-1.10.4.custom.min.js"></script>
    <script>
      $(document).ready(function(){
        $('button').button();
        $('#slider').slider({
          values : [0, 100],
          min : 0,
          max : 100,
          slide: function(event, ui){
            $('.minVal').text(ui.values[0]);
            $('.maxVal').text(ui.values[1]);

            $('#minValSelected').val(ui.values[0]);
            $('#maxValSelected').val(ui.values[1]);
          }
        });
      });
    </script>
  </body>
</html>
```

We included the jQuery UI theme CSS file in the `head` section. Then, we have some CSS rules for the elements on the page. Other classes and their properties are the same as the previous CAPTCHA. We defined a new CSS class for the element with the `id slider` and have set its width.

The document body begins by opening a form with the class `frmCAPTCHA`. Inside the form, there is an `h3` element and two `div` elements with the class name `row`. Inside `h3`, we have displayed both the numbers that have to be set on the slider. First `div` with class `row` has another `div` with `id` as `slider` and two `label` elements, one label before and one after the slider. These labels will hold the current value of slider while it is dragged. The `slider div` will be converted into an actual jQuery UI slider. The last `div` with the class `row` has two hidden fields inside it and a `submit button`. The `id` and `name` attributes of these hidden inputs are `minValSelected` and `maxValSelected`, respectively. Whenever the slider is dragged, the current minimum and maximum values of the slider will be set inside these hidden fields. This finishes up our `form`, after which we include the jQuery js file and jQuery UI js file.

Then, there is the jQuery code that makes the slider functional. Inside the `$(document).ready` handler, we converted the `button` element to jQuery UI `button` component.

Now, we can focus on the slider. Since we have two numbers that are going to be from 0 to 100, we will need 2 handles where each handle will be used to set one number. To create handles, we need to provide the `values` option. It must be an array where the value of each element is set as the handle value. So if you provide three elements in this array, the slider will have three handles.

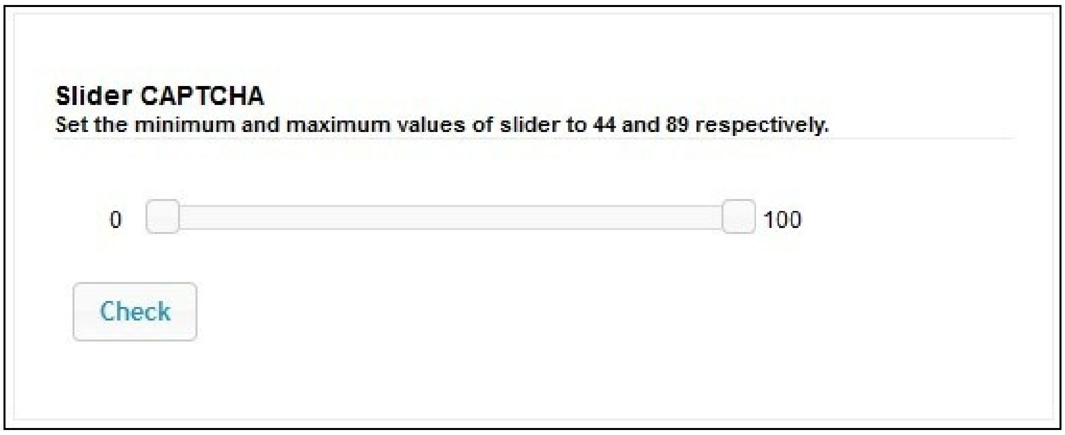
We provided `[0, 100]` as the value for the `slider` because our minimum and maximum values are 0 and 100, respectively, and we need handles for these two. Next, we have set the minimum and maximum values for slider. These values specify the starting and end limits of the slider.

In the end, we defined the `slide` method. This method is invoked when a slider handle is being dragged. Like other jQuery UI methods, this method receives two parameters: `event` and `ui`. The `ui` object is what we need here. This object has three properties:

- `handle`: This is the object corresponding to the slider handle that is being dragged
- `value`: This is the current value of the current handle
- `values`: This is the an array that contains the current values of all handles

Using the `ui.values` array, we first update the text of labels that are to the left and right of the slider to the current values of the slider handles. These same values are also being set as the values of the hidden input elements. So, each time any of the slider handles is moved, the hidden input fields will have the updated value.

Our slider is functional now and we can see it in action. Run the `sliderCAPTCHA.php` file using your web server, and you will see a page like this:



Slider CAPTCHA

Set the minimum and maximum values of slider to 44 and 89 respectively.

The page will display which values have to be set on the slider.

Validating the slider values

Clicking on the **Check** button on the page will submit the page, but will do nothing as of now. To check whether the CAPTCHA has passed or failed, we need to validate the values against the ones stored in session.

Open the `sliderCAPTCHA.php` file again and add the validation code in PHP. Add the following code after the line `session_start()`:

```
if(isset($_POST['submit']))
{
  if($_POST['minValSelected'] != $_SESSION['sliderMin'] ||
$_POST['maxValSelected'] != $_SESSION['sliderMax'] )
  {
    ?>
      <div class="row ui-widget" style="line-height: 20px;">
        <div class="ui-state-error ui-corner-all">
          <p>
            <span class="ui-icon ui-icon-alert" style="float: left; margin-
right: .3em;"></span>
            CAPTCHA Failed. Try again.
          </p>
        </div>
      </div>

    <?php
  }
  else
  {
    ?>
      <div class="row ui-widget" style="line-height: 20px;">
        <div class="ui-state-highlight ui-corner-all">
          <p>
            <span class="ui-icon ui-icon-alert" style="float: left; margin-
right: .3em;"></span>
            CAPTCHA Passed.
          </p>
        </div>
      </div>

    <?php
  }
}
```

The first `if` condition is checking whether the form was submitted or not. Inside it, we check the values of the hidden input elements submitted from forms that are in the variables `$_POST['minValSelected']` and `$_POST['maxValSelected']`. The `$_POST['minValSelected']` variables is being checked against the minimum, that is, `$_SESSION['sliderMin']`, and `$_POST['maxValSelected']` is being checked against the maximum, that is, `$_SESSION['sliderMax']`. If either of these values do not match, we display a jQuery UI themed error message; on successful match, we display a success message on the page.

Improving the slider CAPTCHA

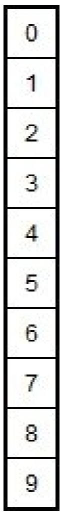
You can make more than two handles and ask the user to set different values for each of them. Instead of numbers, create an array of animal names and select two random names. Now, set up the slider so that the animal name changes on each step. Ask the user to set the slider in such a way that both handles have the required values for the generated names.

Creating the number CAPTCHA

So far, we have seen two examples with draggable, droppable, and slider. For our last implementation, we will use the sortable component of jQuery UI.

We will generate a five-digit number on the server side and display it to the user. Then, we will display five digits of the same numbers in random order and ask the user to arrange them using sortables to make the original number. However, this time there will be a difference in how we display the original number. Instead of printing the five-digit number directly on the screen, we will display it as an image. To do this, we will need an image of all 10 digits from zero to nine. This can be created as a single sprite of 10 images. We will then create five div elements, set the background image to the sprite image, and appropriately calculate and apply the background position property to display the correct number. You can find this image in the code bundle. Look inside the `Chapter5` folder for an image named `sprite.png`.

Here is what the image looks like:



You will also have to create a new file for this example. Create a new file named `numberCAPTCHA.php` inside the `Chapter5` folder and place the `sprite.png` image here.

Generating the five-digit number

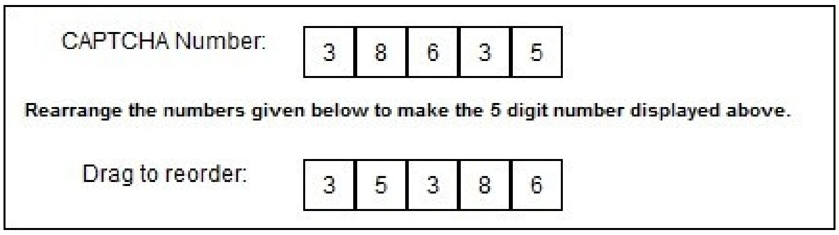
We have used PHP's `rand` function in the previous examples. It will be used here as well to generate a five-digit number. Write the following code in the `numberCAPTCHA.php` file:

```
<?php
  session_start();
  $randomNumber = (string)rand(10000, 99999);
  $_SESSION['CAPTCHAValue'] = $randomNumber;
?>
```

By now, you must be familiar with `session_start`. A new session is started and in the next line, we used the `rand` function to generate a number between `10000` and `99999` and stored it in the variable `$randomNumber`. Note that we have cast the number as a `string`. This has been done because we will be using it as a string array later on. In the last line, this value is stored in a session variable called `CAPTCHAValue`.

Displaying CAPTCHA on the page

We will now design the page where the generated number will be displayed. Along with the generated number, we will display the five digits of this number separately. The following diagram shows the page design we want to create:


CAPTCHA Number:
3 8 6 3 5
Rearrange the numbers given below to make the 5 digit number displayed above.
Drag to reorder:
3 5 3 8 6

The following code needs to be added after the closing tag of PHP:

```
<html>
  <head>
    <meta charset="utf-8">
    <title>Number CAPTCHA</title>
    <link rel="stylesheet" href="css/ui-lightness/jquery-ui-
1.10.4.custom.min.css">
    <style type="text/css">
      body{
        font-family:arial,verdana;
        font-size:12px;
        margin: 0px auto;
        width: 500px;
      }

      #frmCAPTCHA{
        border: 1px solid #EEEEEE;
        float: left;
        margin: 0 auto;
        padding: 20px;
        width: 100%;
      }

      h3{
        border-bottom:1px solid #eee;
      }

      .row{
        display: block;
        padding: 10px;
        clear: left;
      }
      .row label{
        float: left;
        padding: 0px 10px;
        width: 125px;
        text-align: center;
      }
      .bgNumber{
        background: url("sprite.png") no-repeat scroll 0 0 rgba(0, 0, 0,
0);
        display: block;
        float: left;
        height: 27px;
        width: 27px;
      }

      #CAPTCHATiles{
        float:left;
      }
      .clear{
        clear:both;
      }
    </style>
  </head>
  <body>
    <form id="frmCAPTCHA" method="post">
      <h3>Number CAPTCHA</h3>

      <div class="row">
        <label>CAPTCHA Number: </label>
        <?php
          $arrayNumbers = array();
          for($i =0; $i<5; $i++)
          {
            array_push($arrayNumbers, $randomNumber[$i]);
            $pos = ($randomNumber[$i] * 26 * -1);
        ?>
        <div class="bgNumber" style="background-position:0px <?php echo
$pos;?>px;"></div>
        <?php
      }
      ?>
      </div>

      <div class="row">
        <small><strong>Rearrange the numbers given below to make the 5
digit number displayed above.</strong></small>
      </div>

      <div class="row">
      <?php
        shuffle($arrayNumbers);
      ?>
      <label>Drag to reorder: </label>
      <div id="CAPTCHATiles">

        <?php
          for($i =0; $i<count($arrayNumbers); $i++)
          {
            $pos = ($arrayNumbers[$i] * 26 * -1);
            echo '<div data-value="'.$arrayNumbers[$i].'"
name="letter_'.$arrayNumbers[$i].'" id="letter_'.$arrayNumbers[$i].'"
class="bgNumber" style="background-position:0 '.$pos.'px;"></div>';
```

```
          }
          echo '</div>';
        ?>
      </div>

      <div class="row">
        <input type="hidden" name="filledCAPTCHAValue"
id="filledCAPTCHAValue"/>
        <button type="submit" name="submit">Check</button>
      </div>
    </form>
    <script src="js/jquery-1.10.2.js"></script>
    <script src="js/jquery-ui-1.10.4.custom.min.js"></script>
  </body>
</html>
```

As we did earlier, we included the jQuery UI CSS file in the `head` section and added some CSS rules after it. Since the numbers will be displayed as individual divs, we have created a CSS class named `bgNumber` that will set the background image as `sprite.png` and will also set the `height` and `width` of the element. Other CSS classes and properties have also been defined to fix the look of the other elements.

Coming to the `body` section, there is a `form` with `id` `frmCAPTCHA`. Inside it, there is an `h3` element that serves as the page heading. After this, there is a `div` having the class `row` where we display the CAPTCHA value, that is, our five-digit number.

To display the number, a `for` loop is used that runs through the length of the string. Each iteration of the loop gives us access to a digit of the five-digit string. This digit is multiplied by 26 and then multiplied by -1 and the result is stored in the variable `$pos`. In the next line, a `div` with the class `bgNumber` is created and its `background-position` CSS property is set. The `background-position` property requires setting two values for the *x* and *y* axes of the background image of the element where “0px 0px” is “horizontal vertical”. In our sprite image, each digit covers around 26 pixels space horizontally and vertically. Hence, we are multiplying the digit by 26. Multiplying it with -1 makes this value negative. So, if 0 is top, any negative value will be towards the bottom of the image. Since the image is vertical, we do not need to change the *x* axis position for the background. The value in the `$pos` variable is set as the *y* axis position value.

For example, if the digit is 4, it becomes *4 * 26 * -1* which is -104. We set the background position to “0 -104px”. It causes the sprite to position itself on the div at the 104th pixel position vertically, which displays the sprite where the number written is 4.

Moving on, in the next line, we display an informative message for users. There is another `div` with the `class` `row` inside where we have to display the shuffled numbers. To shuffle the five-digit string that we have, we use the in-built `str_shuffle` function of PHP and store the result in the variable `$shuffledNumber`. Then, inside this `div`, we create another `div` with `id` `CAPTCHATiles`. We now loop on the variable `$shuffledNumber` inside this `div` and set the `background-position` property as we did earlier.

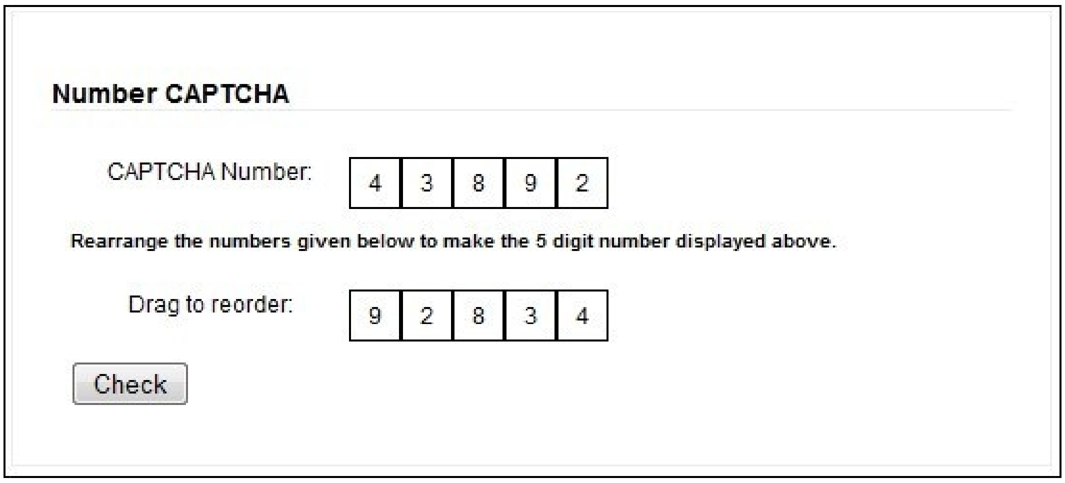
Note

We have an extra `data-value` attribute while creating divs here. The value for this attribute is the digit itself. This value will be used just before the form is submitted to create the five-digit number the user has made.

We have now two sets of five-digit numbers. The first one represents the original number that was generated on the server, and the second set is the same five digits but in random order.

The final `div` with the `class row` contains a hidden input field and a submit button. The hidden field has its `id` and `name` as `filledCAPTCHAValue`. This field will hold the five-digit number that the user has made after arranging the numbered boxes.

Close the `form` tag and include the jQuery JavaScript and jQuery UI JavaScript files after it. If you run the file on your browser now, you will see both the original five-digit number and the shuffled digits under it. Reloading the page will result in different numbers each time.


Number CAPTCHA
CAPTCHA Number: 4 3 8 9 2
Rearrange the numbers given below to make the 5 digit number displayed above.
Drag to reorder: 9 2 8 3 4
Check

Adding the sortable functionality

Until now, users were not able to rearrange the divs to make the number required for CAPTCHA. For this, we need to make the shuffled digits sortable. After jQuery UI is included, use the following code to make the digits sortable:

```
<script>
  $(document).ready(function(){
    $('button').button();
    $('#CAPTCHATiles').sortable({
      cursor : 'pointer'
    });

    $('#frmCAPTCHA').submit(function(){
      var str = '';
      $('#CAPTCHATiles div.bgNumber').each(function(){
        str+= $(this).data('value');
      });
      $('#filledCAPTCHAValue').val(str);
    });
  });
</script>
```

In the first line, we initialized the button component. After that, we made all the `bgNumber` elements inside the `div CAPTCHATiles` sortable. We have only provided an option here, which is the `move` value for the `cursor` property that will change the cursor as sorting starts.

When the user clicks on the **Check** button, we need to know which five-digit number the user has made after arranging the sortable boxes. To achieve this, we find all the sortable divs, that is, the divs with the class `bgNumber` and get the value of their data attributes. A variable `str` is appends all these values together, which will give us a five-digit string representing the number that the user has made with the sortables. We assign this number as a value to the hidden element `filledCAPTCHAValue`.

Validating the number on the server

Clicking on the **Check** button will send the value of the element `filledCAPTCHAValue` to the server, where it will be available in the variable `$_POST['filledCAPTCHAValue']`. We can check this value against the value in session to see if the user has arranged the numbers correctly. The following PHP code will be appended to the file `numberCAPTCHA.php` just after the line that has the call to the `session_start()` function:

```
if(isset($_POST['submit']))
{
  if($_POST['filledCAPTCHAValue'] != $_SESSION['CAPTCHAValue'])
  {
    ?>
      <div class="row ui-widget" style="line-height: 20px;">
        <div class="ui-state-error ui-corner-all">
          <p>
            <span class="ui-icon ui-icon-alert" style="float: left; margin-
right: .3em;"></span>
            CAPTCHA Failed. Try again.
          </p>
        </div>
      </div>
    <?php
  }
  else
  {
    ?>
      <div class="row ui-widget" style="line-height: 20px;">
        <div class="ui-state-highlight ui-corner-all">
          <p>
            <span class="ui-icon ui-icon-alert" style="float: left; margin-
right: .3em;"></span>
            CAPTCHA Passed.
          </p>
        </div>
      </div>
    <?php
  }
}
```

The preceding code is pretty straightforward. We check the value `$_POST['filledCAPTCHAValue']` against the value in the session, which is `$_SESSION['CAPTCHAValue']`. In the case of a successful match, we display a success message, or display an error message on failure.

Summary

We used three different jQuery UI components in this chapter. Although not many options for components were used, these examples should give you the knowledge required to use these components in unusual ways. Our purpose is to not only learn about the jQuery UI components, but to learn about their practical usages and the ability to think out of the box.

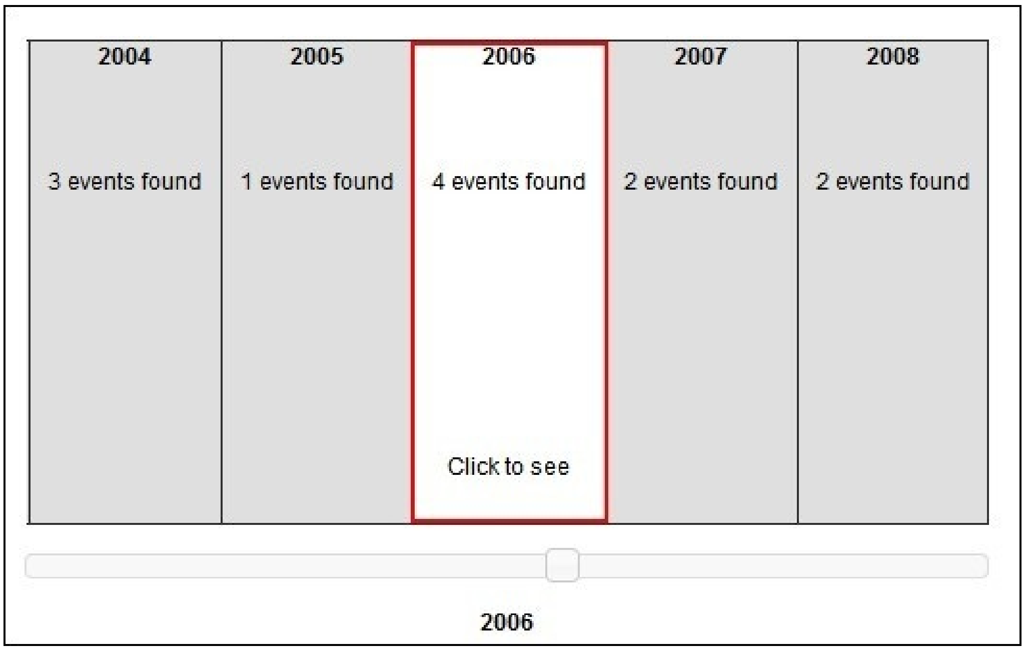
In the next chapter, we will make extensive use of the slider component to create an event timeline. Meanwhile, try to think differently and create CAPTCHAs using other components.

Chapter 6. Creating an Event Timeline Using a Slider

We have used many components of the fabulous jQuery UI so far. In the previous chapter, Chapter 5, *Implementing CAPTCHA Using Draggable and Droppable*, we used draggable, droppable, and the slider functionalities to create some new CAPTCHA implementations. In this chapter, we will use the slider again, and we will create an event timeline that makes use of it.

The following screenshot shows what the timeline will look like. It will be a scrollable, horizontal list of events from the year 2001 to the year 2010. The data for the events of each year will be kept in a JavaScript object and displayed using JavaScript.

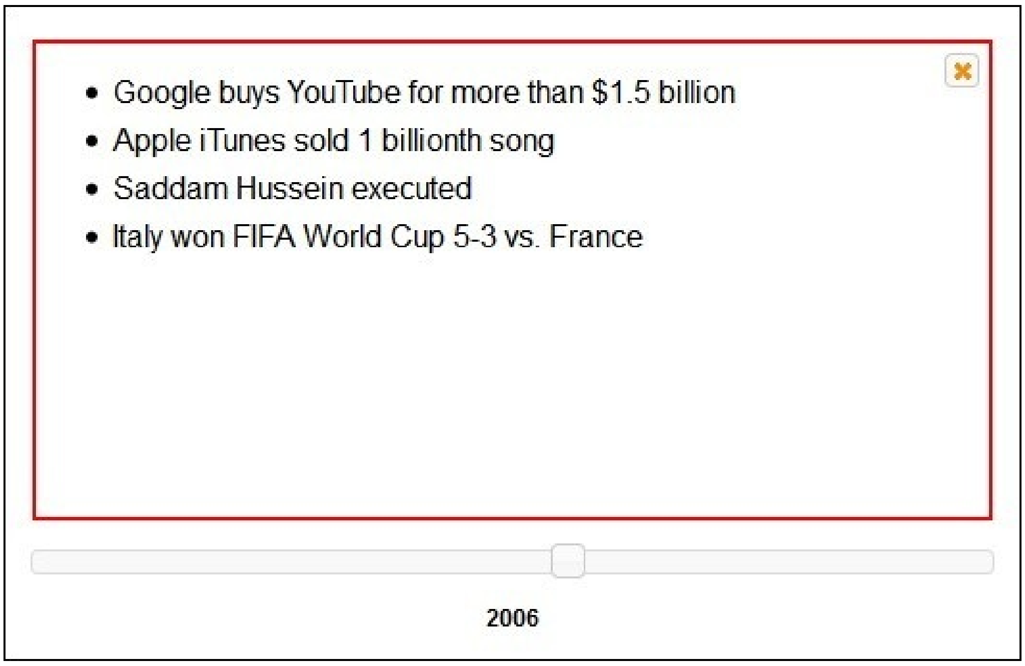
Each year will be displayed as a separate block wherein we will display the number of events that occurred in that year. Initially, five blocks for each year will be visible when the page loads, but we will keep the number of items to display configurable:



There will be a slider under the timeline. It will have its range from 2001 to 2010. Dragging the slider will scroll the timeline to set focus to the year set by the slider. If the selected year is not visible, the timeline will be scrolled to bring it in focus.

To focus on a particular year, we will use another `div` element as a window. In the preceding screenshot, we can see **2006** selected by a red-bordered window. The user will be able to drag this window to set focus on other years. This will also set the slider to that particular year.

Clicking on any year will expand the corresponding div block to full width and will list all events in detail, which will resemble the following screenshot:


- Google buys YouTube for more than $1.5 billion
- Apple iTunes sold 1 billionth song
- Saddam Hussein executed
- Italy won FIFA World Cup 5-3 vs. France

2006

There will be some mathematics required to perform the animation, and calculations to position the elements dynamically, so grab a cup of coffee and jump to the next section.

Creating the folder structure

The first step is to create the required folder structure:

1. Just as in the previous chapters, create a folder named `Chapter6` inside the `MasteringjQueryUI` folder.
2. Inside this folder, create an HTML file and name it `index.html`. This file will keep our HTML markup.
3. Also copy the `js` and `css` folder inside the `Chapter6` folder.
4. Now go inside the `js` folder, create a new file, and name it `timeline.js`. This file will have all of the JavaScript code required for our timeline.

We can now move to the next step and write the HTML markup.

Designing the page

HTML markup and CSS are very important when it comes to designing a feature like timeline using a slider, because precision will be required to position the elements. Most of the data will be filled using JavaScript. Because of this, we will need minimal markup, but it must be accurate. Write this HTML code in your `index.html` file to create a basic structure:

```
<html>
  <head>
  <meta charset="utf-8">
  <title>Event Timeline</title>
  <link rel="stylesheet" href="css/ui-lightness/jquery-ui-
1.10.4.custom.min.css">
  </head>
  <body>
    <div class="container">
      <div id="timeline"></div>

      <div id="leftOverlay"class="overlay" ></div>

      <div id="rightOverlay" class="overlay" ></div>

      <div id="window">
        <div class="ui-state-default ui-corner-all close"><span class="ui-
icon ui-icon-closethick"></span></div>
        <div id="yearEvents"></div>
        <div class="link">Click to see</div>
      </div>

    </div>

    <div class="clear"> </div>
    <div id="slider"></div>
    <div class="clear">&nbsp;</div>
    <label id="sliderVal"></label>

    <script src="js/jquery-1.10.2.js"></script>
    <script src="js/jquery-ui-1.10.4.custom.min.js"></script>
    <script src="js/timeline.js"></script>
  </body>
</html>
```

We start by creating a `div` element with a `class container`. This `div` will contain the entire markup for the timeline. Inside it are four more div elements. The first element has its `id` set as `timeline`. This `div` will contain individual div elements corresponding to each year. There are 10 years in our case (from 2001 to 2010). Then there are two `div` elements with their `id` values as `leftOverlay` and `rightOverlay`, respectively. If you recall the first screenshot earlier in this chapter, you will realize that the year **2006** is selected and other years to its left and right have overlays on them. Therefore, `leftOverlay` will act as an overlay for the elements that are towards left of the currently selected year, and `rightOverlay` will be the overlay for the elements to the right of currently selected year.

At the end is a `div` with its `id` set as `window`. This `div` will act as a handle or window for the selected year. User will be able to drag it and set focus to any other year. Clicking it will expand it to cover whole `timeline div` and events will be listed inside it. To achieve this, we will require three div elements inside it.

The first `div` is a close icon that has been created using the `ui-icon-closethick` class of the jQuery UI themeroller. Clicking on it will collapse the window and return the timeline to its original state. The second `div` has the `id` set as `yearEvents`. It will display all the events for a selected year. The final `div` is just for information. It asks the user to click when the window is focused on a particular year.

Since markup is incomplete without any CSS styling, we will now write the CSS rules for different elements.

Styling the content

We will have to be very precise about the CSS rules for all elements, especially for all the elements inside the `div` having `id` `container`. Write the following CSS rules in the `head` section of the `index.html` file, and then we will go through each of these to see the concepts behind them:

```
<style type="text/css">
  body{
    font-family:arial,verdana;
    font-size:12px;
    margin: 0px auto;
    width: 900px;
  }
  .container{
    border: 1px solid #333;
    border-left:0;
    height: 250px;
    margin: 100px auto 0;
    overflow: hidden;
    position:relative;
    width: 500px;
  }
  #timeline{
    border-left: 1px solid;
    width: 2000px;
    position:absolute;
  }
  .year{
    border-left: 1px solid #333;
    float: left;
    height: 250px;
    width: 99px;
    text-align:center;
  }
  .year div{
    margin-top: 50px;
  }
  #window       {
    border: 2px solid #ff0000;
    height: 247px;
    left: 0;
    position: absolute;
    cursor:pointer;
    top: 0;
    width: 99px;
    z-index:300;
  }
  .close{
    display:none;
    position: absolute;
    right: 5px;
    text-decoration: underline;
    top: 5px;
  }
  .link{
    bottom: 20px;
    position: absolute;
    text-align: center;
    width: 100%;
  }
  .overlay{
    background-color: #2b2b2b;
    height: 250px;
    opacity: 0.15;
    position:absolute;
    z-index:100;
  }
  #leftOverlay{
    left:0;
  }
  #rightOverlay{
    right:0px;
    width: 400px;
  }
  #yearEvents ul li{
    font-size: 16px;
    line-height: 25px;
  }
  #slider{
    margin:0 auto;
    width:500px;
  }
  #sliderVal{
    width: 100%;
    display: block;
    text-align: center;
    font-weight:bold;
  }

  .clear{
    clear:both;
  }
</style>
```

Now let's try to understand the important properties of all the CSS rule declarations one by one:

- `body`: General CSS properties that will be applied to whole document are defined here. The font details and document width are declared. The document is centered.
- `.container`: This class defines the outer body of the `timeline`. We have applied all borders except on the left. The left border will be applied by a `year div`. The three most important properties to note here are `width`, `position`, and `overflow`. Since the `div` for one year is going to be `100px` wide and we are initially going to display 5 years, we have set the width to `500px`. Since the overlays and the `window div` inside this `div` will have to be positioned absolutely, we have set the `postion` property for the `container` as `relative`. This will cause the inside elements to position them relative to the `container div`. Because the `width` is set to only `500px` and there can

be more than five `div` elements for years, we have set the `overflow` to `hidden`. This will hide all elements after the `500px` `width`.

- `#timeline`: This `div` will contain all other divs for each year, so we have defined its `width` as `2000px`.
- `.year`: This `class` will be applied to `div` elements for individual years. We have defined its `width` as `99px`. The `1px` value has been assigned to left border. A `height` of `250px` has also been assigned. This will give us the appearance of the columns.
- `.year div`: This rule will be applied to `div` elements inside the year `div`.
- `#window`: This `div` will work as a focus window for a year. To make it stand out, we have applied a `2px` red-colored border to it. It has also been positioned `absolute`, and the `left` margin has been set to `0`. Absolute positioning is required so that we can easily position it anywhere inside the `timeline div`. Its `z-index` value has been defined as `300` to make it appear on top.
- `.close`: This `div` will work as a close icon when the window is expanded. Initially, it will be hidden, so we have set its `display` property to `none`. It has been positioned absolutely and `5px` far from the `top` and `right` margins.
- `.overlay`: We will have two overlays, one on each side of the `window div`. This class defines general CSS properties for them. We have set the `background-color` and `opacity` to make it semi-transparent. The `height` has been set to `250px`, and `absolute` positioning has been done so that we can adjust the positions whenever the `window div` moves.
- `#leftOverlay`: This is specific to the overlay on the left of `window div`. We have set its `left` property to `0` because initially, the window will be focused on the first year of the `timeline`, and we do not need left overlay. Since its parent is the `container div`, it will be positioned relative to the container.
- `#rightOverlay`: This overlay has been positioned to the right and its `width` is `400px`. This is easy to calculate as the `width` of `window div` is set to `100px`, which leaves a 400 px space to its right for the overlay. Since its parent is the `container div`, it will be positioned relative to the container.
- `#yearEvents ul li`: When a particular year is clicked on, we will display all events for that year in the form of an unordered list `ul`. This list will be appended to the `div` with the `id` `yearEvents`. We have set the `font-size` and `line-height` properties for each list item `li` in the list.
- `#slider`: Similar to the `timeline`, the `slider` `width` has been made `500px` and set to the center of the document using the `margin` property.
- `#sliderVal`: This is a label where we will display the currently selected year when the slider is sliding.
- `.clear`: This is the generic CSS class used to clear the floats.

With the preceding list, we have covered all the CSS rules required to structure the `timeline` div as well as to beautify it to some extent.

Save the `index.html` file if you have not done so and open it in browser. The basic structure of the page will resemble the following screenshot:

Since no jQuery has been applied, the slider will not be visible yet. The rest of the display will be created using jQuery and jQuery UI.

Let us now go to the next section, where we will jump into jQuery and get our JavaScript code structure ready to implement the timeline.

Getting the code structure ready

Just as we saw in the last couple of chapters, we will continue with the object-literal pattern to organize our code. Our `$(document).ready()` section will have only a call to the `init` method of the object, which will encapsulate the rest of the functionality. We will first declare all the properties and methods required to make the timeline functional, and later implement each method.

Navigate to the `js` folder in your filesystem and open the `timeline.js` file. In this file, write the following code to set up our object and a `$(document).ready()` handler to call its `init` method:

```
$(function(){
  objTimeline.init();
});
var objTimeline =
{
  itemsToDisplay : 5,
  minYear : 0,
  maxYear : 0,
  currentYear : 0,
  maxScrollYear : 0,
  timelineWindowStartYear : 0,
  windowLeft:0,
  isWindowOpen : false,
  timelineData :
  [
    {
      year : 2001,
      events : ['Human Genome Sequence Revealed', 'World Economic
Slowdown']
    },
    {
      year : 2002,
      events : []
    },
    {
      year : 2003,
      events : ['Space shuttle Columbia crashed', 'India and Pakistan reach
cease-fire in Kashmir', 'Earthquake in Iran kills over 15,000 people']
    },
    {
      year : 2004,
      events : ['NASA rover Opportunity lands on Mars', 'Yasar Arafat
dies', 'Bird flu spreads in many countries']
    },
    {
      year : 2005,
      events : ['Hurricane Katrina on August 29']
    },
    {
      year : 2006,
      events : ['Google buys YouTube for more than $1.5 billion', 'Apple
iTunes sold 1 billionth song', 'Saddam Hussein executed', ' Italy won FIFA
World Cup 5-3 vs. France']
    },
    {
      year : 2007,
      events : [' Halo 3 released', 'Microsoft released Windows Vista']
    },
    {
      year : 2008,
      events : ['Barack Obama became first African-American president of
USA', 'Summer olympic games held in Beijing']
    },
    {
      year : 2009,
      events : ['Israel attacks Gaza', 'Michael Jackson dies at 50 years']
    },
    {
      year : 2010,
      events : ['Apple released first iPad', 'Earthquake in Haiti']
    }
  ],
  init : function()
  {
    this.createMarkup();
    this.createTimeline();
  },
  createMarkup : function()
  {
  },
  createTimeline: function()
  {
  },
  closeWindow : function()
  {
  }
};
```

The first block of code is jQuery's event handler, which is fired once the page has loaded and the DOM is ready. Inside it, we call the `init` method of an `objTimeline` object. Let's look at each of the properties of `objTimeline` in detail:

- `itemsToDisplay`: This is a configuration option with which we will decide how many years have to be displayed in the timeline. This parameter is very important as we will have to calculate the size of `container div` using this parameter only. Initially, we have set it to `5`.
- `minYear`: From the list of events, we will calculate the minimum year that is available for events and store it in this property.
- `maxYear`: Just like `minYear`, this property will store the year with the maximum value. Both the `minYear` and `maxYear` properties will be used to set the minimum and maximum values for the slider.
- `currentYear`: This stores the year that is currently selected either by the slider or by dragging the `window div`.
- `maxScrollYear`: This property will be calculated at runtime. It defines the maximum

year beyond which the timeline will not scroll. Let's see why this is important. We have a 500 px wide `container` and years from 2001 to 2010. When the `timeline div` scrolls and 2006 is in the extreme left, we have 2010 as the last available year on the right. Now, if we select 2007, the timeline must stay intact and only the focus window should move. Otherwise, we will have only 4 years visible in the timeline, and empty space to the right.

- `timelineWindowStartYear`: This property will store the value of the year that is currently at the extreme left of the timeline. Along with the previous property, `maxScrollYear`, the decision to scroll or not to scroll the window will be taken here.
- `windowLeft`: This will store the value of the CSS left property for the `div` with the `id window`. Since clicking on a year window will expand it to take up the full width and closing it will collapse it to the original size, we need to know which position it was in before expanding.
- `isWindowOpen`: This keeps track of whether a year window is open in the expanded view or not. Initially, it will be `false`.
- `timelineData`: This property is where we define the year and events for those years. As you can see in the code, `timelineData` is an array of objects. Each object in the array represents an array and has two properties, namely `year` and events. The `events` property is an array where each item of the array represents an event. In our case, we defined years and their events from 2001 to 2010.
- `init`: This is the first method of `objTimeline` to be called. It simply calls two more methods, `createMarkup` and `createTimeline`.
- `createMarkup`: This method will be responsible for creating markup for the `timeline div` and the `years div` from the `event` array defined via the `timelineData` property.
- `createTimeline`: Event handlers for slider, dragging window div, window click, and so on will be implemented here.
- `closeWindow`: This is used to close the `window div` when it is expanded after clicking on a specific year.

The preceding object structure makes clear the way we are going for our implementation. We will start by creating and displaying the timeline on our page.

Creating the timeline markup from data

To create the required markup, we will iterate on the array defined for the `timelineData` property and fill the `div` with the `id` `timeline`. We will also resize the `container` `div` based on the value of the `itemsToDisplay` property. Other properties such as `minYear`, `maxYear`, `currentYear`, `timelineWindowStartYear`, and `maxScrollYear` will also be set.

In the `timeline.js` file, locate the `createMarkup` method and write this code to create the timeline:

```
$('.container').css({width: (objTimeline.itemsToDisplay*100)+'px'});
$('#rightOverlay').css({ width: ((objTimeline.itemsToDisplay * 100) - 100)
+ 'px' });
this.minYear = this.timelineData[0].year;
this.maxYear = this.timelineData[0].year;
var strYearDivs = '';
for(var i=0; i< this.timelineData.length; i++)
{

  strYearDivs+= '<div class="year">';
  strYearDivs+= '<strong>'+ this.timelineData[i].year + '</strong>';
  strYearDivs+= '<div class="numEvents">' +
(this.timelineData[i].events.length) + ' events found</div>';
  strYearDivs+= '</div>';
  this.minYear = this.timelineData[i].year < this.minYear ?
this.timelineData[i].year : this.minYear;
  this.maxYear = this.timelineData[i].year > this.maxYear ?
this.timelineData[i].year : this.maxYear;

}
this.currentYear = this.minYear;
this.timelineWindowStartYear = this.currentYear;
$('#sliderVal').text(this.currentYear);
this.maxScrollYear = this.maxYear - (objTimeline.itemsToDisplay - 1);
$('#timeline').html(strYearDivs);
```

The first line sets the `width` of the `div` with the `id` set as `container`. Since we have set `itemsToDisplay` to `5`, a `500px` `width` will be set for the container. Similarly, the next line sets the `width` for the `rightOverlay` `div`. Since we want the first year to be selected (or focused) by default, the width for `rightOverlay` will be 100 px less than the container size.

The next two lines set a default value for `minYear` and `maxYear` by getting the value of the year of the first element in the `timelineData` array.

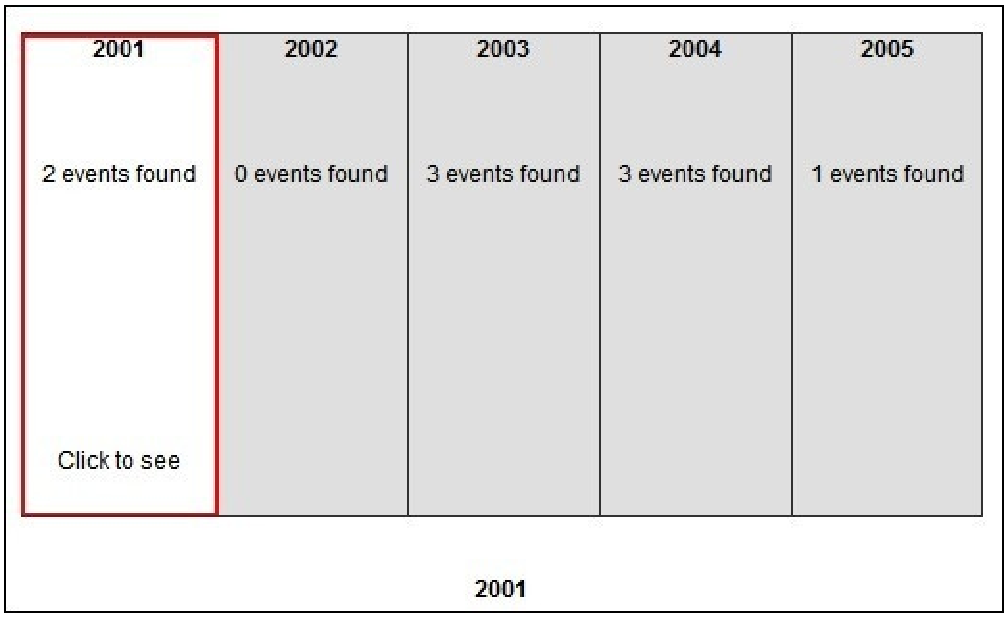
A variable named `strYearDivs` is defined as an empty string. It will hold the DOM for the timeline divs having class `year`. A `for` loop iterates in `timelineData` and creates a `div` with the `year` class, with the year number written inside it. Another `div` with the `numEvents` class is created, in which the length of the `events` array for each array element is written. This loop also sets the correct `minYear` and `maxYear` values using two ternary operators.

After the loop is complete, we have the HTML string ready as well as the correct values for `minYear` and `maxYear`.

Now we set the `currentYear` property to `minYear`. The `timelineWindowStartYear` property is set to `currentYear` as well because the first year will be focused by default. The HTML of the `sliderVal` label is also set to `currentYear`.

The penultimate line sets the `maxScrollYear` property. If you recall, `maxScrollYear` is the value of the year beyond which the timeline will not scroll. It is calculated by subtracting `itemsToDisplay-1` from `maxYear`. For our example, `maxYear` is 2010 and `itemsToDisplay` is 5, which gives us `maxScrollYear` as `(2010 – (5-1)) = 2006`. Thus, 2006 will be the value beyond which the timeline will not scroll.

In the last line, we insert the `strYearDivs` HTML string into the `timeline` div. At this moment, you can save the file and see after reloading the page in the browser that 5 years have appeared in timeline and the window is fixed on the first year, as shown in the following screenshot:



You can change the value of the `itemsToDisplay` property and see that the UI changes accordingly.

Implementing the timeline functionality

The page that we designed is static as of now. There are three things we need to change to get the timeline working: slider functionality, window dragging, and displaying events for selected year.

Let us begin implementing them one by one, starting with the slider.

Making the slider work

The slider will have its range set to the `minYear` and `maxYear` properties. We will update the value of the `sliderVal` label as the slider slides, and when it stops, we will animate the interface that includes the positioning of the `timeline`, `window div`, `leftOverlay div` and `rightOverlay div`. In order to achieve this, we will write the following code inside the `createTimeline` method:

```
$('#slider').slider(
{
  min: objTimeline.minYear,
  max: objTimeline.maxYear,
  step : 1,
  start : function(event, ui)
  {
    if(objTimeline.isWindowOpen)
    {
      objTimeline.closeWindow();
    }
  },
  slide: function(event, ui)
  {
    objTimeline.currentYear = ui.value;
    $('#sliderVal').text(objTimeline.currentYear);
  },
  stop : function(event, ui)
  {
    if(objTimeline.currentYear >= objTimeline.maxScrollYear)
    {
      objTimeline.timelineWindowStartYear = objTimeline.maxScrollYear;
      //animate timeline
      $('#timeline').animate(
      {
        left : (objTimeline.timelineData.length -
objTimeline.itemsToDisplay) * 100 * -1
      }, 400);

      var yearsToScroll = objTimeline.currentYear -
objTimeline.maxScrollYear;

      //animate window
      $('#window').animate(
      {
        left :yearsToScroll * 100
      }, 400);

      //animate overlays
      $('#leftOverlay').show().animate(
      {
        width: (yearsToScroll * 100)
      }, 400);
      $('#rightOverlay').show().animate(
      {
        width: ((objTimeline.itemsToDisplay -1) * 100 ) -
yearsToScroll * 100 )
      }, 400);
    }
    else
    {
      objTimeline.timelineWindowStartYear = objTimeline.currentYear;
      var yearDiff = Math.abs(objTimeline.currentYear -
objTimeline.minYear);
      var newLeft = ((yearDiff * 100)) * -1;
      $('#timeline').animate(
      {
        left : newLeft
      }, 400);

      $('#window').animate(
      {
        left : 0
      }, 400);

      $('#leftOverlay').hide();

      $('#rightOverlay').show().animate(
      {
        width: (objTimeline.itemsToDisplay * 100 ) - 100
      }, 400);
    }
  }
});
```

We begin by initializing the jQuery UI slider and defining its properties. First, the `min` and `max` properties are set to `minYear` and `maxYear`, respectively, which will set the lower and upper bounds for the slider to slide.

Next, we set the `step` property to `1`. This property determines the value by which the slider will increment or decrement on each slide. Its value is 1 by default, so even if we do not provide the step property, it will work perfectly for our application. It can be useful in certain cases; for example, when you want to step decade by decade, you can set step to 10.

After properties, we defined the `start`, `slide`, and `stop` methods. The `start` method is called when the user starts sliding, `slide` is called when the slide is in progress, and `stop` is called when the sliding stops.

Inside the start method, we check the value of the `isWindowOpen` property. If it is set to true, we call the `closeWindow` method. Since `isWindowOpen` has been set to false initially, if block will not be executed. The `isWindowOpen` method will be set to `true` later when the user clicks on a year and the window expands to cover whole timeline. We are calling `closeWindow` method on startup because we want the timeline to be in its original state when the sliding starts.

Since the `slide` method is called when the slider is being moved, we are setting the value for `currentYear`. The `ui` parameter receives the current value in the `ui.value` property. This will ensure that we have an updated value whenever the slider moves.

The `stop` method is where we scroll the timeline and set the positions for the window and overlays based on the selected year. This is a bit complex, so please go through it slowly.

The if block

We have an `if` block that will be executed only if the value of currently selected year is equal to or more than `maxScrollYear`. Since `maxScrollYear` is 2006 in our case, the `if` block will be executed only if `currentYear` is 2006 or greater.

Since in this case timeline will not scroll but only windows and overlays will scroll, we set the `timelineWindowStartYear` property to `maxScrollYear`. Now we set the `left` property of the timeline so that the div that represents the `maxScrollYear` year remains on the extreme left. We have used jQuery's animate function, which will set the left property by animating it in `400` milliseconds. The formula we are using is `(objTimeline.timelineData.length - objTimeline.itemsToDisplay) * 100 * -1`. This will give us `(10 - 5) * 100 * -1`, which is *-500*. The negative value will cause the `timeline div` to scroll to the left and set 2006 in the extreme left.

Note

Remember that in this case, 2006 will always be on the extreme left and the timeline will not scroll (this applies if `itemsToDisplay` is equal to `5`).

Now we need to animate the window and the overlays. We have calculated the difference in number of years in the `yearsToScroll` variable.

To set up the window, we need to set its left property depending on which year is selected. If 2006 is selected, its left will property be 0; if 2007 is selected, left will be 100; left will be 200 for 2008; and so on. Hence, the formula becomes `yearsToScroll * 100`.

To set the `leftOverlay div`, we need to set its `width`. Again, if the year is 2006, it should not be visible, or width can be 0; in the case of 2007, the width will be 100 px. Therefore, the same formula (`yearsToScroll * 100`) will be applied. Since `leftOverlay` is absolutely positioned, its left value is set to 0 and it will cover the timeline from left to right.

For `rightOverlay`, the formula becomes `((objTimeline.itemsToDisplay -1) * 100 ) - (yearsToScroll * 100 )`. Since `rightOverlay` begins from the right, it will cover the timeline until it touches the right border of the window.

Since width and left are all numerical values, we have used jQuery's animate function to create the visual effect when windows and overlays change their position.

The else block

The `else` block will be executed when we need to scroll the timeline as well. The calculations here are simple compared to those of the `if` block.

In this case, the selected year will always be the first item in the `timeline div`, so we start by setting `timelineWindowStartYear` to `currentYear`. Then we calculate the absolute number of years between `minYear` and `currentYear`. For example, if `minYear` is 2001 and `currentYear` is 2005, the difference is 5 years. Since each `year div` is 100 px wide, we can set the left property of the timeline to 500 px, which will position the `div` for `currentYear` to the extreme left.

For `leftOverlay` can simply hide it because the `currentYear div` is already on the extreme left, and we do not need a left overlay.

The calculation of the width of `rightOverlay` is pretty straightforward as well. We can use the `(objTimeline.itemsToDisplay * 100 ) – 100` formula to find the width. Using the `itemsToDisplay` property rather than hard coding a value such as 5 or 3 is a good idea. It gives you the flexibility to configure more.

This concludes our scrolling and animations for the slider. After saving the file, reload the page in your browser and give the slider a try. You will see the `div window` and the left and right overlays moving with smooth animation.

Dragging the year window

Apart from the slider, we also want to be able to drag the `window div` in the visible viewport to focus on other years. This should also move the overlays and the slider value. For this, we will convert the `window div` into a draggable and use its methods to control the UI behavior. In the same method after the slider code, write this code to make the window draggable and move overlays:

```
$('#window').draggable(
{
  containment: '.container',
  grid : [100,0],
  cursor: 'pointer',
  drag : function(event, ui)
  {
    var leftPos = ui.position.left;
    $('#leftOverlay').css({width: leftPos}).show();
    $('#rightOverlay').css({width : (objTimeline.itemsToDisplay * 100) -
leftPos - 100}).show();
  },
  stop : function(event, ui)
  {
    var leftPos = ui.position.left;
    leftPos = leftPos/100;
    objTimeline.currentYear = objTimeline.timelineWindowStartYear +
leftPos;
    $('#slider').slider('value', objTimeline.currentYear);
    $('#sliderVal').text(objTimeline.currentYear);
  }
});
```

The first option is `containment`, which has been set to `.container`. Since the `#window` div is inside `div.container`, we don't want it to go outside while dragging. Instead of specifying `.container`, we could write the parent as well.

The `grid` option is used to snap the grid by an (x,y) distance horizontally and vertically. We specified it as (100,0), which means that dragging the div horizontally will move it 100 px away from its current position.

The `cursor` property changes the cursor while the div is being dragged. We specified pointer as its value.

Next, we implemented the `drag` method. This method invokes while div is being dragged. Since the window is being dragged and the draggable component will position it, we need to take care of the left and right overlays. The drag event receives the left CSS property as a property of the `ui` parameter. We store it in the `leftPos` variable. Then we set the width of `leftOverlay` to `leftPos`. Doing so expands `leftOverlay` from the extreme left to the left border of the window.

Similarly, we set the width of `rightOverlay` using the `(objTimeline.itemsToDisplay * 100) - leftPos - 100})` formula.

Once the user stops dragging the window, the `stop` method of draggable is called. Inside this method, we first take the left CSS value for the window as in the `drag` method. Because each `year div` is 100 px wide, we divide it by 100 to find out how many years it has moved by. The number of these years is added to `timelineWindowStartYear` to get the year where the window is currently placed. Once we have calculated the value, we set the value of the `currentYear` property. After this, we set the slider's value to `currentYear` and change the text inside the `sliderVal` label to reflect the currently selected year.

If you reload the page in the browser now, you will be able to drag the window. Overlays will adjust their widths accordingly, and once you stop dragging, the slider will also be set to the year where the `window div` rests.

Displaying event details when a year window is clicked on

With all the sliding and dragging done, we now have to add the `click` event, which works when a `year div` is clicked. It will expand the window to cover the full timeline and we will display the event details inside it:

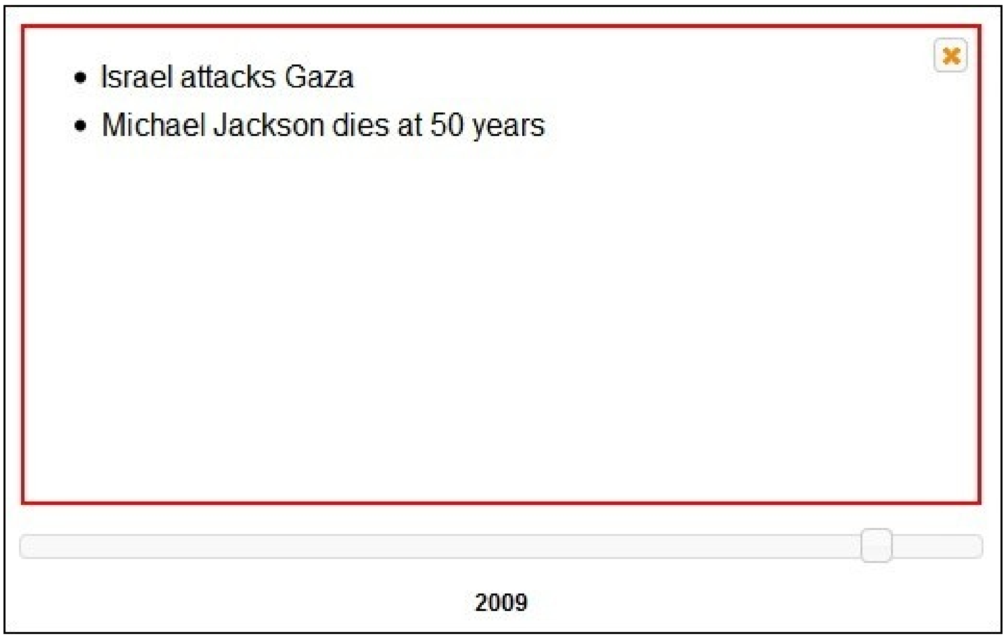
1. Write this code after the window dragging code to implement the click handler for `div#window`:

```
$('#window').click(function()
{
  if(objTimeline.isWindowOpen)
  {
    return;
  }
  objTimeline.isWindowOpen = true;
  $('.link').hide();
  objTimeline.windowLeft = $(this).css('left');
  $(this).css({'background-color' : '#fff'})

  .animate({
    left : 0,
      width : (objTimeline.itemsToDisplay * 100 ) -4 + 'px',
      height: '246px'
  }, 100, function()
  {
    $('.container').css({'border' : 0});
    $('.close').show();
    var str = '<ul>';
    for(var i=0; i <objTimeline.timelineData.length; i++)
    {
      if(objTimeline.timelineData[i].year == objTimeline.currentYear)
      {
        var allEvents = (objTimeline.timelineData[i]).events;
        if(allEvents.length == 0 )
        {
          str+= '<li>No events found.</li>';
        }
        for(var j=0; j< allEvents.length; j++)
        {
          str+= '<li>';
          str+= allEvents[j];
          str+= '</li>';
        }
        break;
      }
    }
    str+= '</ul>';
    $('#yearEvents').html(str);
  });
});
```

2. Check whether the window is already open using `isWindowOpen` property. If window

is already expanded for some other `year`, we just return from the handler.

3. Otherwise, we set `isWindowOpen` to `true`.
4. Then we hide the `div` having the `link` class.
5. Next we set the value of the `objTimeline.windowLeft` property by assigning it the current CSS to `left` for the `div` having the `id` as `window`. We need to keep this property because we will need to reset it after the window is closed.
6. Now we set the background color of the window to white and animate it by setting the `left`, `width`, and `height` properties. The `left` property is set to `0` and `width` is set to full width of the `container div` so that the window covers the full `timeline div`. For the animation, we provide a time of 100 ms and a callback function that fires when the animation is completed.
7. After the animation is completed, we set the border of the parent container to 0 and display the close icon.
8. Now we proceed to create an unordered list of events for the selected year. This is done easily by iterating on the `objTimeline.timelineData` array and comparing the year value of each array item with the `currentYear` property.
9. Once a match is found, we create the HTML of the list's items by iterating in the events array for that year.
10. Once complete, this HTML is inserted into the `yearEvents` element.
11. This can be checked by reloading the page and clicking on a year. If you clicked on 2009, you will see something like this:



- Israel attacks Gaza
- Michael Jackson dies at 50 years

Closing the event details window

Finally, we are left with implementing the close button and resetting the timeline to its original state:

```
$('.close').click(function(event)
{
  objTimeline.closeWindow();
});
```

The event handler for the close button calls the `closeWindow` method of `objTimeline`. If you remember, we called the `closeWindow` method in the `start` method of `slider` as well. Go to the `closeWindow` method in your file and write this code to close and reset the window:

```
$('.container').css({'border' : '1px solid #333', 'border-left' : 0});
$('#yearEvents').empty();
$('.close').hide();
$('.link').show();
$('#window').animate(
{
  width: '99px',
  left : objTimeline.windowLeft
}, 500, function()
{
  $(this).css({'background-color' : 'transparent'});
  objTimeline.isWindowOpen = false;
});
```

This is basically some cleanup we are doing. Borders are applied to the `div` `container` as it was originally. Then the `yearEvents` `div` is emptied. The close button is hidden and the `link` `div` is displayed again.

Finally, we `animate` the `window` `div` to reset it to its original size and position. We set its size to `99px`, and the `left` value is set to `windowLeft`, which we had saved earlier. A `500` ms duration is set for the animation, and a callback function is provided.

The callback function sets the `background-color` of window to `transparent` and resets the `isWindowOpen` property to `false` so that it may be used later when another window is configured.

Improving the timeline

Here are a few suggestions on how some more features can be added to the timeline to make it more useful:

- Currently, we are relying on sequential years for some calculations. Design the timeline such that nonsequential years and missing years can be handled (let me give you a hint; use an index).
- Along with the slider, implement a date picker that allows the user to select a date and search for matching events for that specific date.
- Implement jQuery UI's effects to close the expanded window.

Summary

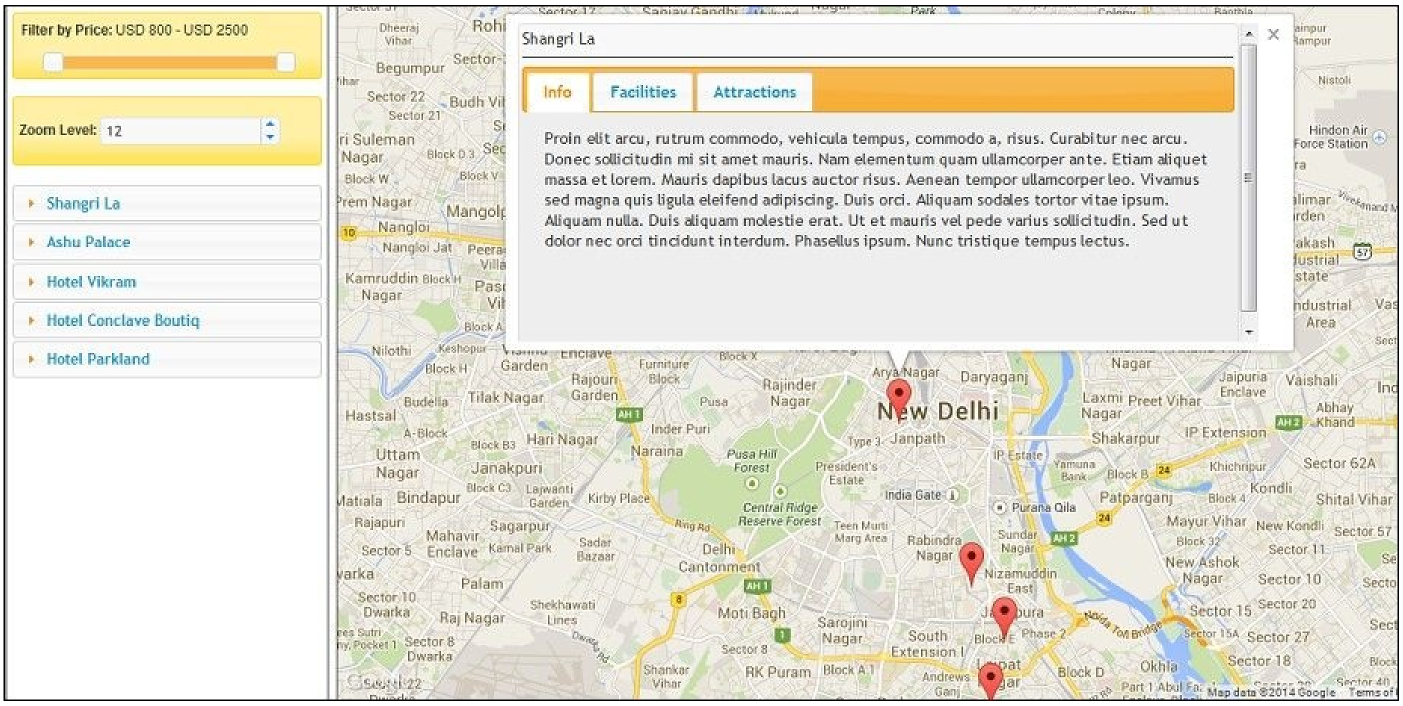
We saw that even the simplest of components, such as slider, can be helpful in creating nice effects when used creatively. We used the slider in a unique way to achieve a nice timeline effect in this chapter. There was a bit of mathematics involved as we did many calculations to position the elements correctly.

In the next chapter, we will explore another jQuery UI component called spinner, along with the slider, to create a Google Maps mashup that lists hotels of a city.

Chapter 7. Using jQuery UI with Google Maps API

We made extensive use of the slider in the previous chapter to display a timeline of events. In this chapter, we will create a mashup using some components of jQuery UI along with the Google Maps API.

We will create a page to display the hotels in a particular city. The page will be divided into two columns. On the left-hand side, we will display the list of hotels, and on the right-hand side, we will have a Google map of the city with locations of hotels marked in the map. Clicking on any marker on the map will display an information window with some more details. The end result will resemble the following screenshot:



Proin elit arcu, rutrum commodo, vehicula tempus, commodo a, risus. Curabitur nec arcu. Donec sollicitudin mi sit amet mauris. Nam elementum quam ullamcorper ante. Etiam aliquet massa et lorem. Mauris dapibus lacus auctor risus. Aenean tempor ullamcorper leo. Vivamus sed magna quis ligula eleifend adipiscing. Duis orci. Aliquam sodales tortor vitae ipsum. Aliquam nulla. Duis aliquam molestie erat. Ut et mauris vel pede varius sollicitudin. Sed ut dolor nec orci tincidunt interdum. Phasellus ipsum. Nunc tristique tempus lectus.

During this process, you will learn the functionalities of the Google Maps API, along with jQuery UI components. The jQuery UI components that we will use in this chapter are slider, accordion, tabs, and spinner.

Creating the folder structure

Just as we did in previous chapters, let’s begin by creating the folder structure. Create a new folder named `Chapter7` inside the `MasteringjQueryUI` folder. Go inside this folder and create an HTML file called `index.html` for our HTML markup. Copy the `js` and `css` folders of jQuery UI inside the `Chapter7` folder. Now go inside the `js` folder, create a new file, and name it `myMap.js`. This file will contain all of the code required for creating our maps mashup.

Getting a Google Maps API key

We will be using Google Maps JavaScript API v3 to display the maps. Google requires that you register your application and get an API key. Getting an API key is pretty easy. Visit https://developers.google.com/maps/documentation/javascript/tutorial#api_key from your browser to get detailed instructions on how to obtain an API key.

Once you are done setting up the folder structure and have an API key, move on to the next step – designing the page.

Designing the page

The left-hand side of the page will have jQuery UI's slider, spinner, and an accordion. The slider will be used to filter hotels based on price, and the spinner will control the zoom level of the map. The accordion will display the hotels such that the header of each accordion panel will display the hotel name and its content panel will have the hotel price and some description text. Write the following markup in the `index.html` file to create the HTML structure:

```
<html>
  <head>
    <meta charset="utf-8">
    <title>Google Maps with jQuery UI</title>
    <link rel="stylesheet" href="css/ui-lightness/jquery-ui-
1.10.4.custom.min.css">
  </head>
  <body>
    <div>
      <div class="left" >
        <div class="ui-state-highlight ui-corner-all" style="padding:
7px;">
          <strong>Filter by Price:</strong>
          <span id="currentRange"></span>
          <div class="clear">&nbsp;</div>
          <div id="slider"></div>
        </div>

        <div class="clear">&nbsp;</div>

        <div class="ui-state-highlight ui-corner-all" style="padding:
5px;">
          <p>
            <strong>Zoom Level:</strong>
            <input id="spinner" value="12" readonly="readonly"/>
          </p>
        </div>

        <div class="clear">&nbsp;</div>

        <div id="listing">
        </div>

      </div>

      <div class="right" id="hotelsMap"></div>

      <div id="tabs" style="display:none;">
        <ul>
          <li><a href="#info">Info</a></li>
          <li><a href="#facilities">Facilities</a></li>
          <li><a href="#attractions">Attractions</a></li>
        </ul>
        <div id="info">
```

```
          Proinelitarcu, rutrumcommodo, vehicula tempus, commodo a,
risus.Curabiturnecarcu. Donecsollicitudin mi sitametmauris. Nam elementum
quam ullamcorper ante.Etiamaliquetmassa et lorem. Maurisdapibus lacus
auctorrisus.Aeneantemporullamcorperleo. Vivamussed magna quis ligula
eleifendadipiscing.Duisorci.Aliquamsodalestortor vitae
ipsum.Aliquamnulla.Duisaliquammolestieerat.Utetmaurisvelpedevariussollicitu
din. Sedut dolor necorcitinciduntinterdum. Phasellusipsum.Nunctristique
tempus lectus.
        </div>
        <div id="facilities">
          <ol>
            <li>High Speed Internet</li>
            <li>Health Club</li>
            <li>Airport pickup and drop</li>
            <li>Bar</li>
            <li>Cultural Activities</li>
          </ol>
        </div>
        <div id="attractions">
          <ol>
            <li>International Airport - 45 minutes/16.00 Kms</li>
            <li>New Delhi Railway Station - 10 minutes/3.00 Kms</li>
          </ol>
        </div>
      </div>
    </div>

    <script type="text/javascript"
src="https://maps.googleapis.com/maps/api/js?key=YOUR_API_KEY_HERE">
</script>
    <script src="js/jquery-1.10.2.js"></script>
    <script src="js/jquery-ui-1.10.4.custom.min.js"></script>
    <script src="js/myMap.js"></script>
  </body>
</html>
```

Inside the `head` section, we have the page title and have included jQuery UI’s CSS theme file.

In the `body` section, there is a wrapper `div` element. Inside this `div`, there is another `div` with the `left` CSS class. This `div` will hold the slider, spinner, and accordion. First we created a div for slider and applied jQueryUI’s `ui-state-highlight` and `ui-corner-all` CSS classes to it to theme it. Inside it is a `strong` element that works as a label. Then there is a `span` element with the `id` value `currentRange` where we will display the price range selected using the slider. Finally comes another `div` with the `id` value `slider`. This `div` will be converted to a slider using the `slider` component of the library.

Next, we cleared the floats and created another `div` for the spinner. We applied the same `ui-state-highlight` and `ui-corner-all` classes to theme it. Inside this `div` are a `strong` element and a text box with the `id` set as `spinner`. We made this text box `readonly` and set its value to `12` so that when the map is loaded, we can set the zoom to `12`. Google maps generally have zoom levels starting from 0 to 18. A value of 12 is enough to focus on most cities.

The last element inside the `div` with the `left` class is a `div` with the `id` value `listing`. This `div` will list all the hotels and will be converted into an accordion.

After `div` with the `left` class, we have another `div` with the `right` class and the `id` value `hotelsMap`. This `div` will work as a canvas and will be used to display the maps.

Finally, we have a div with the `id` set as `tabs` and it's `display` set to `none`. We will convert this into jQuery UI tabs. This `div` will be used after the maps are loaded. When a marker is clicked on, the `infowindow` will open. We will display these tabs in the `infowindow`. In real life, the data in the tabs will, of course, be filled dynamically. This is just an example of how we can incorporate any jQuery UI component in `windows.infowindow`. For this chapter, we have created three tabs named `Info`, `Facilities`, and `Attractions`, each of which will display its content for a particular hotel.

The structure of the tabs adheres to what jQuery UI recommends. There is an unordered list with three `li` elements inside it. Each of these `li` elements has an anchor element with the value of the `href` attribute set.

After the list `ul`, there are three `div` elements. The `id` of each div corresponds to the `href` value provided in `ul`.

Finally, before closing the `body` tag, we included all JavaScript files required. The first is the Google Maps API, which has been included by setting the `src` value as `https://maps.googleapis.com/maps/api/js?key= YOUR_API_KEY_HERE`. Do not forget to replace the value of `key` with your API key. After this, jQuery source file and jQuery UI source files are included. Finally, `myMap.js` is also included.

This is the entire markup that will be required to display the page initially. Let's now style it by applying CSS rules for different elements.

Styling the content

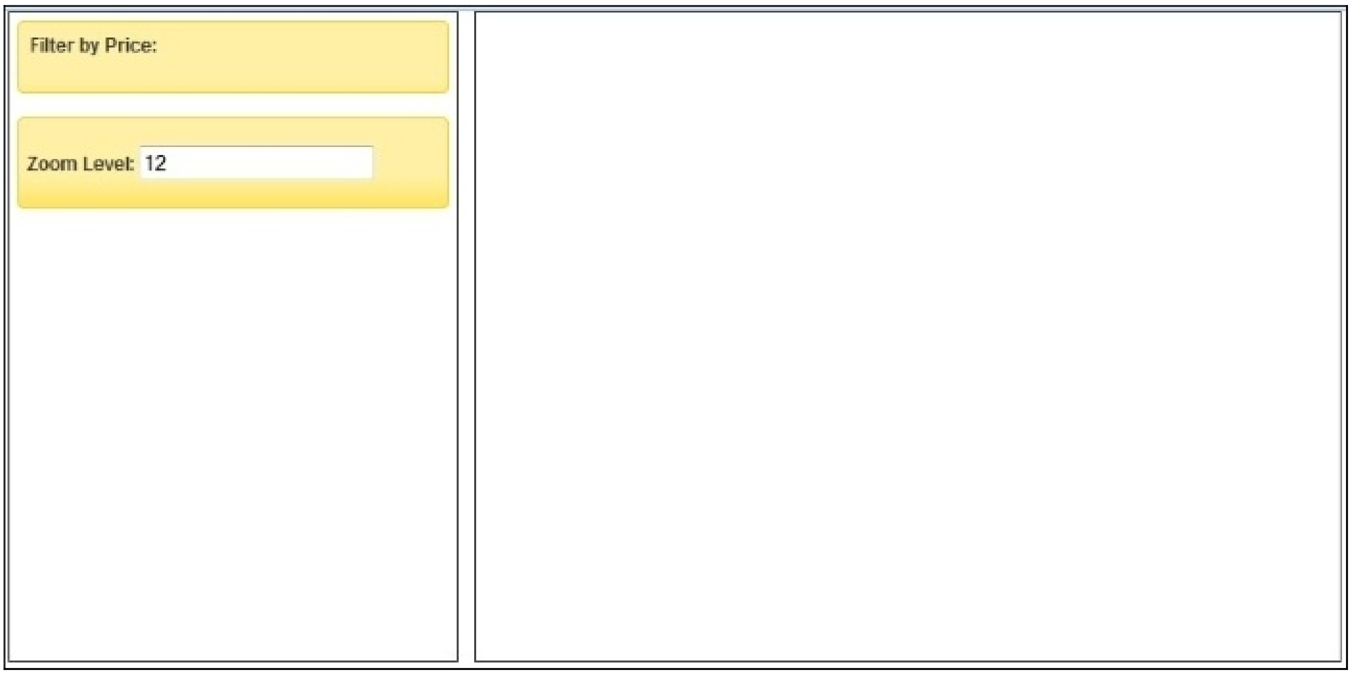
We need to float the div element with the left class to the left and the maps div to the right. We also need separation for the slider, spinner, and accordion. You can write the following CSS rules in the head section after the jQuery UI theme file has been included:

```
<style type="text/css">
  body
  {
    font-family:arial,verdana;
    font-size:12px;
    margin: 0px auto;
    width: 100%;
  }
  .left
  {
    height:600px;
    width:20%;
    border: 1px solid #333;
    float:left;
    padding:5px;
    margin-left:10px;
  }
  .ui-accordion .ui-accordion-content
  {
    padding:10px;
  }

  #slider
  {
    margin-left:10%;
    width:80%;
  }
  .right
  {
    height:610px;
    width:77%;
    border: 1px solid #333;
    float:right;
    margin-right:10px;
  }
  .clear
  {
    clear:both;
  }
</style>
```

There are very few styles that we have defined here. The styles for body set its font properties and center the page. The next style is for the div with the left class. We gave it a fixed height of 600px and 20% width. It is also floated to left and has a border. After this, we override the padding property for the accordion content panels by giving a 10px padding. Styles for slider include only the setup of the margin and width.

Finally, the style for div with the right class includes setting up its height and width. It is also floated to `right`. A `border` has been provided, and a small `margin` is also added.

After saving the `index.html` file, if you load it in browser, you will see a page that resembles the following screenshot:



With the markup and CSS ready, we can now write the JavaScript to make this page live.

Getting the code structure ready

By now, you should have become comfortable with the object literal notation that we have been using. We will work with it in this chapter as well, and first define an object with all the properties we require to create the maps mashup. Open the `myMap.js` file inside the `js` folder and use the following code to create the object structure and initialize it:

```
$(document).ready(function(){
  myMap.initialize();
});

var myMap =
{
  map : null,
  markers : [],
  infowindow  : null,
  minPrice : 0,
  maxPrice : 0,
  hotelsList : [
    {
      name : 'Shangri La',
      lat : '28.631541',
      lng : '77.213287',
      price : 1000,
      description: 'Pellentesqueaccumsanmolestieipsumutfeugiat.
Nuncvariusnislsed ligula vehicula, vitae sodales magna volutpat! Praesent
tempus faucibusnisl, velaliquetlectusviverraquis.Curabiturleoenim,
tinciduntviverravestibulumluctus, cursus et velit. Proin id metusut mi
sagittisvarius in at nulla.Aliquam semper
lobortispellentesque.Donecaliquamrisus sit ametipsumconsecteturpulvinar.',
    },
    {
      name : 'Ashu Palace',
      lat : '28.652257',
      lng : '77.19243',
      price : 1893,
      description: 'Pellentesqueaccumsanmolestieipsumutfeugiat.
Nuncvariusnislsed ligula vehicula, vitae sodales magna volutpat! Praesent
tempus faucibusnisl, velaliquetlectusviverraquis.Curabiturleoenim,
tinciduntviverravestibulumluctus, cursus et velit. Proin id metusut mi
sagittisvarius in at nulla.Aliquam semper
lobortispellentesque.Donecaliquamrisus sit ametipsumconsecteturpulvinar.'
    },
    {
      name : 'Hotel Vikram',
      lat : '28.573668',
      lng : '77.245388',
      price : 2500,
      description: 'Pellentesqueaccumsanmolestieipsumutfeugiat.
Nuncvariusnislsed ligula vehicula, vitae sodales magna volutpat! Praesent
tempus faucibusnisl, velaliquetlectusviverraquis.Curabiturleoenim,
tinciduntviverravestibulumluctus, cursus et velit. Proin id metusut mi
sagittisvarius in at nulla.Aliquam semper
lobortispellentesque.Donecaliquamrisus sit ametipsumconsecteturpulvinar.'
```

```
    },
    {
      name : 'Hotel Conclave Boutiq',
      lat : '28.556124',
      lng : '77.241197',
      price : 2361,
      description: 'Pellentesqueaccumsanmolestieipsumutfeugiat.
Nuncvariusnislsed ligula vehicula, vitae sodales magna volutpat! Praesent
tempus faucibusnisl, velaliquetlectusviverraquis.Curabiturleoenim,
tinciduntviverravestibulumluctus, cursus et velit. Proin id metusut mi
sagittisvarius in at nulla.Aliquam semper
lobortispellentesque.Donecaliquamrisus sit ametipsumconsecteturpulvinar.'
    },
    {
      name : 'Hotel Parkland',
      lat : '28.588139',
      lng : '77.23526',
      price : 800,
      description: 'Pellentesqueaccumsanmolestieipsumutfeugiat.
Nuncvariusnislsed ligula vehicula, vitae sodales magna volutpat! Praesent
tempus faucibusnisl, velaliquetlectusviverraquis.Curabiturleoenim,
tinciduntviverravestibulumluctus, cursus et velit. Proin id metusut mi
sagittisvarius in at nulla.Aliquam semper
lobortispellentesque.Donecaliquamrisus sit ametipsumconsecteturpulvinar.'
    },
  ],
  initialize : function()
  {
    this.setMinMaxPrices();
    this.displayHotels();
    this.setSpinner();
    this.createMap();
    this.setMarkersAndInfoWindow();
    this.setSlider();
  },
  setMinMaxPrices : function()
  {
  },
  displayHotels : function()
  {
  },
  setSpinner : function()
  {
  },
  createMap : function()
  {
  },
  setMarkersAndInfoWindow : function()
  {
  },
  setSlider : function()
  {
  }
};
```

The event handler for `$(document).ready()` calls the `initialize` method of the `myMap` object. Since this event handler is called only after the page is loaded, it is safe to call the `initialize` method even though we are going to define it later.

In the `myMap` object, let's first look at each of its properties in a nutshell, and then we will implement each of the methods:

- `map`: This will store the reference to the Google maps object that will be created.
- `markers`: This is used to keep track of all the markers placed in the map, markers has been declared as an array.
- `infowindow`: On clicking on a marker, an `infowindow` will be opened. Since only one `infowindow` can be open at any time, we need to store its reference so that we can open it when other markers are clicked on.
- `minPrice`: This is the lowest available price from the list of hotels.
- `maxPrice`: This is the highest available price from the list of hotels.
- `hotelsList`: This is the main array that stores information about hotels we will display in the left panel. In real-world cases, this can be populated using JSON data from the backend. For this chapter, we have created the array with five elements (that is, five hotels). Each hotel has five properties: `name`, `lat`, `lng`, `price`, and `description`, where `lat` and `lng` refer to latitude and longitude, respectively, and will be used to place markers on the map. For the example in this chapter, all the coordinates are of hotels in New Delhi, India.
- `initialize`: This is the entry point for the object. This method is responsible for calling other methods of the object one by one. We have already called upcoming six methods in this method.
- `setMinMaxPrices`: This is a simple method to set the values of the `minPrice` and `maxPrice` properties.
- `displayHotels`: Using the `hotelsList` property, this method will populate the `div` with `id listing` and convert it into an accordion.
- `setSpinner`: We will initialize the `spinner` component in this method and write an event handler that will allow us to zoom into the map.
- `createMap`: As the name suggests, this method will simply initialize the map using Google Maps.
- `setMarkersAndInfoWindow`: After the map is displayed, this method will place markers on the map. It will also have event handlers that will open `infowindow` when a marker is clicked on.
- `setSlider`: This will convert the `div` with the `id slider` into a jQuery UI `slider` component and will set its range between `minPrice` and `maxPrice`. We will also write event handlers that will show or hide hotels from the accordion based on the price. Markers will also be displayed or hidden depending on the values set by the slider.

Setting minimum and maximum prices

To find out the lowest and highest prices available, we will have to loop in the `hotelsList` array and set the values accordingly. The following code will set the minimum and maximum prices for us:

```
setMinMaxPrices : function()
{
  this.minPrice = this.hotelsList[0].price;
  this.maxPrice = this.hotelsList[0].price;
  for(var i = 0; i<this.hotelsList.length; i++)
  {
    this.minPrice = this.hotelsList[i].price <this.minPrice ?
this.hotelsList[i].price :this.minPrice;
    this.maxPrice = this.hotelsList[i].price >this.maxPrice ?
this.hotelsList[i].price :this.maxPrice;
  }
  $('#currentRange').text('USD '+ this.minPrice + ' - ' + 'USD ' +
this.maxPrice);
}
```

We begin by setting both `minPrice` and `maxPrice` as the value of the `price` attribute of the first array element. Then we start a for loop and use the ternary operator to set the values for `minPrice` and `maxPrice`. After the loop finishes, we have both the values set.

We then display these values inside the `span` element with the `currentRange` ID.

Displaying hotels in accordion

To display hotels, we simply have to iterate in the `hotelsList` array and build a DOM. Once this is done, we can push this DOM into the `div` with the `id listing`. After that, we will set the slider as well. Here is the code you can use for the `displayHotels` method:

```
displayHotels : function()
{
  var str = '';
  for(var i = 0; i<this.hotelsList.length; i++)
  {
    var hotel = this.hotelsList[i];
    str+= '<h3 data-price="'+ hotel.price+'">'+hotel.name+'</h3>';
    str+= '<div>';
    str+= '<div class="ui-state-highlight ui-corner-all" style="padding:
5px;">Price: USD ' + hotel.price + '</div>';
    str+= hotel.description;
    str+= '</div>';
  }
  $('#listing').html(str);
  $('#listing').accordion(
  {
    collapsible: true,
    active : false,
    heightStyle : 'content'
  });
}
```

We begin by declaring `str`, a blank string. Then we start the loop. Since we have to convert this hotels list into an accordion, we need to create the HTML compatible with it. We create an `h3` element and a `div` for each hotel. The hotel's name is written inside the `h3` element, and inside the `div`, we write the price of hotel and the description.

Note

Note that we have created a data attribute named `data-price` with the h3 element. It will be used later in the chapter to filter hotels when the slider will be changed.

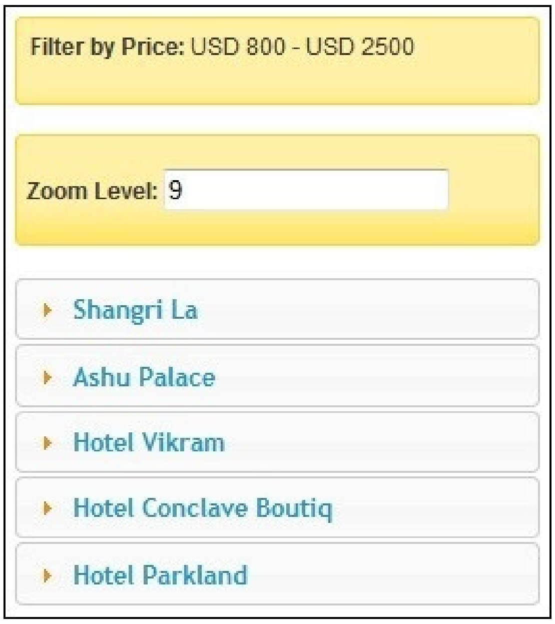
Once the loop is over, we insert this HTML into the `div` with the `id listing`.

Finally, we convert it into an `accordion` component. Note that we have provided three properties while initializing the accordion. These properties are as follows:

- `collapsible`: At least one panel is open as per the accordion's default behavior. If we want to make all panels collapsible at once, this option has to be set to `true`.
- `active`: This property decides which panel to open when the accordion loads. We have set it to `false`, which means that no panel will be open by default. You can also pass a zero-based index if you want to open a particular panel by default. Keep in mind that for `active : false` to work, you should have `collapsible` set to `true`.
- `heightStyle`: We set the `heightStyle` property to `content`, which will make each accordion panel as tall as its content. Two other values are also allowed, which are `auto` and `fill`. When set to `auto`, the height of the accordion is set to the panel with

maximum height. Setting it to `fill` makes the height of accordion equal to its parent container's height.

You can now check out the fruits of hard work we have done by saving the file and viewing it in browser, as shown in the following screenshot:


Filter by Price: USD 800 - USD 2500

Zoom Level: 9

You will see the price range on the top of the left panel, followed by the input element for spinner. After the spinner, there will be the accordion. All panels of the accordion will be closed, and the hotel name will be visible in accordion header. You can click on any hotel name and the content panel will be displayed, which will show the price and some description.

Setting up the spinner

The spinner is a simple component that is great to use with numbers. It has up and down buttons, using which the numerical value of spinner can be increased or decreased.

We will use the `spinner` component to zoom into the map. The following code will create the `spinner` and zoom into the map:

```
setSpinner : function()
{
  $('#spinner').spinner(
    {
      min : 0,
      max : 18,
      stop : function(event, ui )
      {
        myMap.map.setZoom(parseInt($(this).val(), 10));
      }
  });
}
```

We have provided three properties for the `spinner`. The first two are the `min` and `max` values, which we have set to `0` and `18`, respectively. These will restrict the user from selecting values above or below the set limits.

As mentioned earlier in this chapter, 0 and 18 are used because these are the minimum and maximum zoom levels in Google Maps.

The third method is an event handler called `stop`. This is called when a spin occurs, which means that it will be called each time the user clicks on either the up or the down arrow and the value of the spinner changes. Inside this method, we used the `setZoom` method to set the zoom level. This is a method provided by the Google Maps API that takes an integer as a parameter and sets the zoom to the said level. Since we have the map object stored in the map property of the `myMap` object, we are calling `setZoom` on it. The `parseInt` is used because the value of the spinner will be a string and we need to convert it to a number.

Note

Always use the second parameter while using parseInt. The second parameter is the base in which you are trying to parse. Not doing so may result in erratic behavior.

Displaying the map

This method simply creates the map and stores its reference in `myMap.map`. We will center it on New Delhi and set its zoom level to the value set in the spinner. The following code will define the options that we will pass to the map, and then call the maps API to display the map:

```
createMap : function()
{
  var mapOptions =
  {
    center: new google.maps.LatLng(28.637926, 77.223726),
    zoom: parseInt($('#spinner').val(), 10),
    disableDefaultUI : true,
    mapTypeId: google.maps.MapTypeId.ROADMAP,
    scrollwheel: false
  };
  this.map = new google.maps.Map($("#hotelsMap")[0], mapOptions);
  this.infowindow = new google.maps.InfoWindow();
}
```

We created an object named `mapOptions`, where we defined the default values for some properties. Here are the properties we defined:

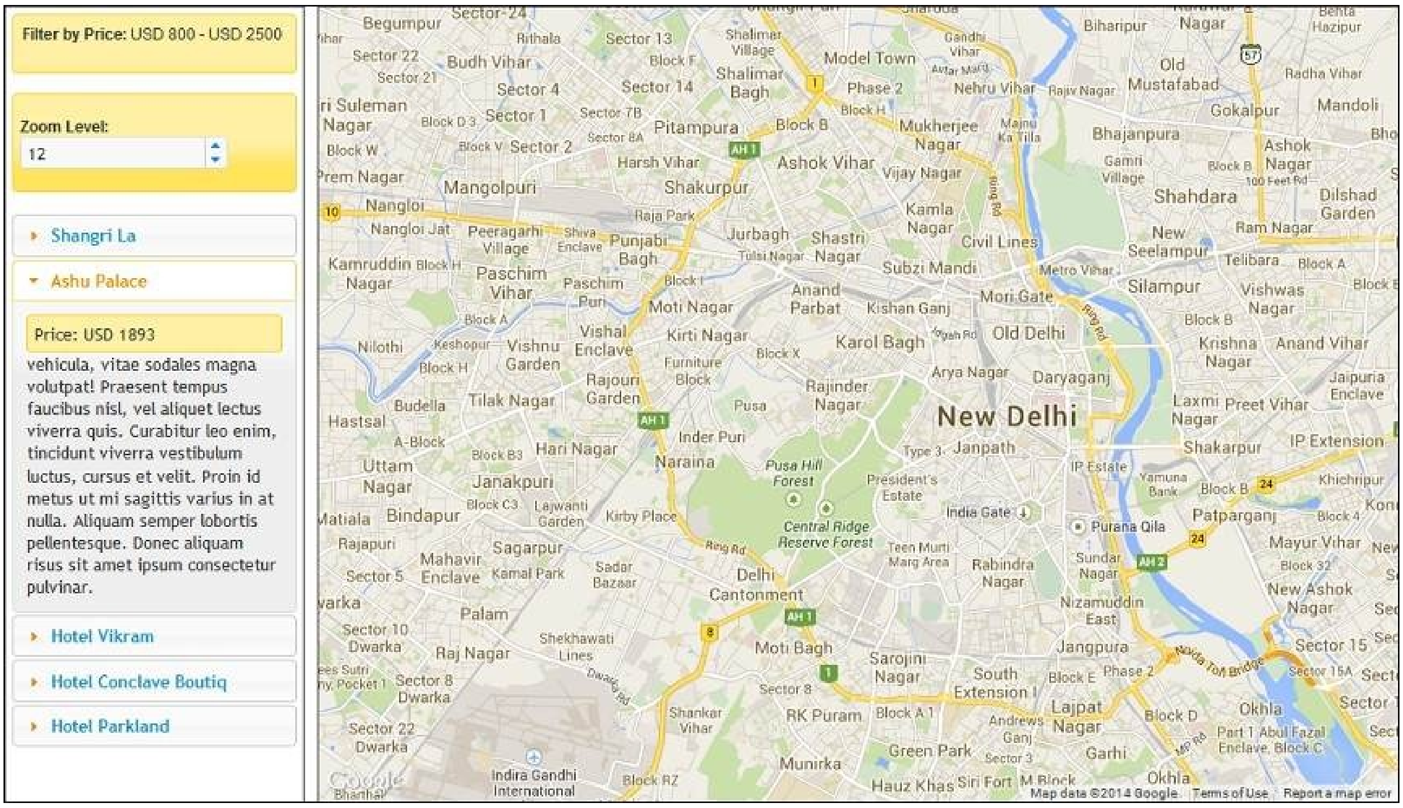
- `center`: The Google maps API's `LatLng` class is used to set the map's center at required coordinates. For this, we pass the latitude and longitude to the `LatLng` classes constructor. The coordinates provided in the preceding code are of New Delhi.
- `zoom`: This sets the zoom level. We just set it to the current value in the input box spinner.
- `disableDefaultUI`: For this example, we will remove all the controls that Google maps loads by default. These include `Zoom`, `Pan`, `MapType`, and `Scale` controls. Setting `disableDefaultUI` to `true` loads the maps without any of these controlled. If you want the default UI but with some of the controls, you can turn the other controls off individually; for example, you can set `panControl` and `zoomControl` to `true` or `false` to add or remove these controls, respectively. Refer to https://developers.google.com/maps/documentation/javascript/controls for more information on controls.
- `mapTypeId`: There are four basic map types that are available in the Google Maps API. These are `ROADMAP`, `SATELLITE`, `HYBRID`, and `TERRAIN`. We have set this property's value to `ROADMAP`.
- `scrollwheel`: Setting `scrollwheel` to `false` disables the zooming using the mouse scroll wheel.

After defining these options, we can load the map. This is done using the Map constructor, where we need to provide two arguments. The first is the DOM element in which we want to load the map, and the second is the options object that we defined in the preceding code.

Since `$("#hotelsMap")` will give us the jQuery object for the `hotelsMap` element, we used `$("#hotelsMap")[0]` to get the DOM element to pass to the map constructor. The reference to the loaded map has been stored in the `myMap.map` property as well.

The last line initializes an `infowindow` and stores its reference in `myMap.infowindow`, which will be used whenever we want to do any operation involving `infowindows`.

Save the file and reload the browser. You will see that the map has been loaded and is centered on Delhi, as shown in the following screenshot. Using the spinner, you will also be able to zoom in or out of the map:



Filter by Price: USD 800 - USD 2500

Zoom Level: 12

Shangri La

Ashu Palace

Price: USD 1893

vehicula, vitae sodales magna volutpat! Praesent tempus faucibus nisl, vel aliquet lectus viverra quis. Curabitur leo enim, tincidunt viverra vestibulum luctus, cursus et velit. Proin id metus ut mi sagittis varius in at nulla. Aliquam semper lobortis pellentesque. Donec aliquam risus sit amet ipsum consectetur pulvinar.

Hotel Vikram

Hotel Conclave Boutiq

Hotel Parkland

Setting markers and infowindows in the map

Now that the map is being displayed, let’s put the markers and info windows on the marker click event. The `myMap.setMarkersAndInfoWindow` method will be used for this.

We will iterate in `hotelsList` and set up a `marker` at each location. Along with setting up the `marker`, we will add the event handler for every click on the `marker`, which will open an `infowindow`. Write this code in your `myMap.js` file to create markers and info windows:

```
setMarkersAndInfoWindow : function()
{
  for(var i = 0; i<this.hotelsList.length; i++)
  {
    var hotel = this.hotelsList[i];
    var marker = new google.maps.Marker(
    {
      position: new google.maps.LatLng(hotel.lat, hotel.lng),
      map: myMap.map,
        title: hotel.name
    });
    this.markers.push(marker);

    google.maps.event.addListener(marker, 'click', function(marker, hotel)
    {
      return function()
      {
        var content = $('#tabs').html();
        myMap.infowindow.setContent('<div id="hotelFeatures"
style="height:280px;">'+ hotel.name+ '<hr/>' + content + '</div>');
        myMap.infowindow.open(myMap.map, marker);

      };
    }(marker, hotel));
  }

  google.maps.event.addListener(myMap.infowindow, 'domready', function(){
    $('#hotelFeatures').tabs();
  });
}
```

To create a marker, we have to use the `Marker` class of Google Maps. There are many properties and methods available in the `Marker` class, but we will confine ourselves to the basic properties and methods because they will let us place the markers.

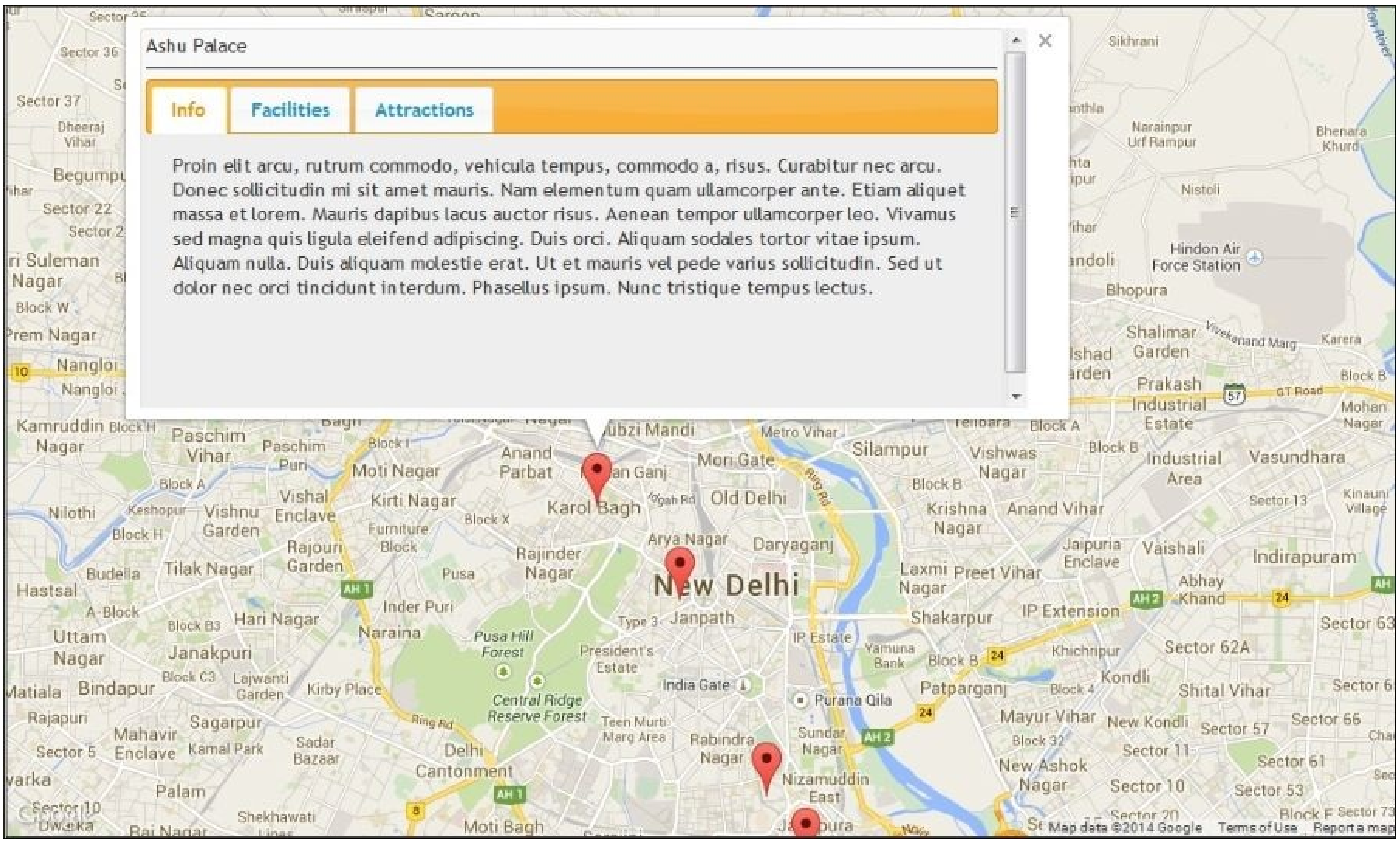
We are looping in the `hotelsList` array and creating a `Marker` per iteration. The three properties we have passed to the constructor of the `Marker` class are `position`, `map`, and `title`. The `position` property refers to the coordinates where the marker will be placed, and like other places, it is represented by the `LatLng` class. The `map` property refers to the map on which we want markers. The last property is `title`, for which we have provided the hotel name `hotel.name` as string. This property appears as a tooltip when the user places the mouse over a `marker`.

We also need to keep track of all the markers so that we can show or hide them when the user changes the price range using the slider. In order to do this, we created an array called `myMap.markers` earlier. Here, we push each marker into this array.

Next, we have to add the event handler on a marker click. This has been done using Google's event handler method, `addListener`, which attached a click event handler to the `click` event of a `marker`. Inside this method, we have a closure being returned by a `return` statement. Inside this closure, we take the HTML of the `div` with `id tabs` in the `content` variable. Using the API's `setContent` method for `myMap.infowindow`, we set the HTML string that will be displayed in the info window. Note that we are wrapping this HTML in a `div` with the `id hotelFeatures`. Finally, we call the `open` method on `myMap.infowindow` to display the info window. The `open` method takes two parameters, which are the map object and the marker object for which `infowindow` is being opened.

After the loop, there is another event listener. It attaches a `domready` event listener to `infowindow`. The event handler will be called each time an info window is created and displayed. Since the `infowindow` contains the HTML structure compatible with a tab component, we convert the `div` with the `id hotelFeatures` to jQuery UI tabs.

Check out the file in the browser again and click on a marker. You will see the tab structure in info windows, as shown in the following screenshot:



Proin elit arcu, rutrum commodo, vehicula tempus, commodo a, risus. Curabitur nec arcu. Donec sollicitudin mi sit amet mauris. Nam elementum quam ullamcorper ante. Etiam aliquet massa et lorem. Mauris dapibus lacus auctor risus. Aenean tempor ullamcorper leo. Vivamus sed magna quis ligula eleifend adipiscing. Duis orci. Aliquam sodales tortor vitae ipsum. Aliquam nulla. Duis aliquam molestie erat. Ut et mauris vel pede varius sollicitudin. Sed ut dolor nec orci tincidunt interdum. Phasellus ipsum. Nunc tristique tempus lectus.

Implementing the slider

We are almost done now. The last bit is to make the slider functional and filter out the hotels from the accordion and markers from the map. We will also use the `slide` method of the `slider` component to change the inner text of the `span` element, which has the `id` `currentRange`. To filter hotels and markers, we will use the `stop` method of the `slider` component.

Here is the code used to create the slider and its events:

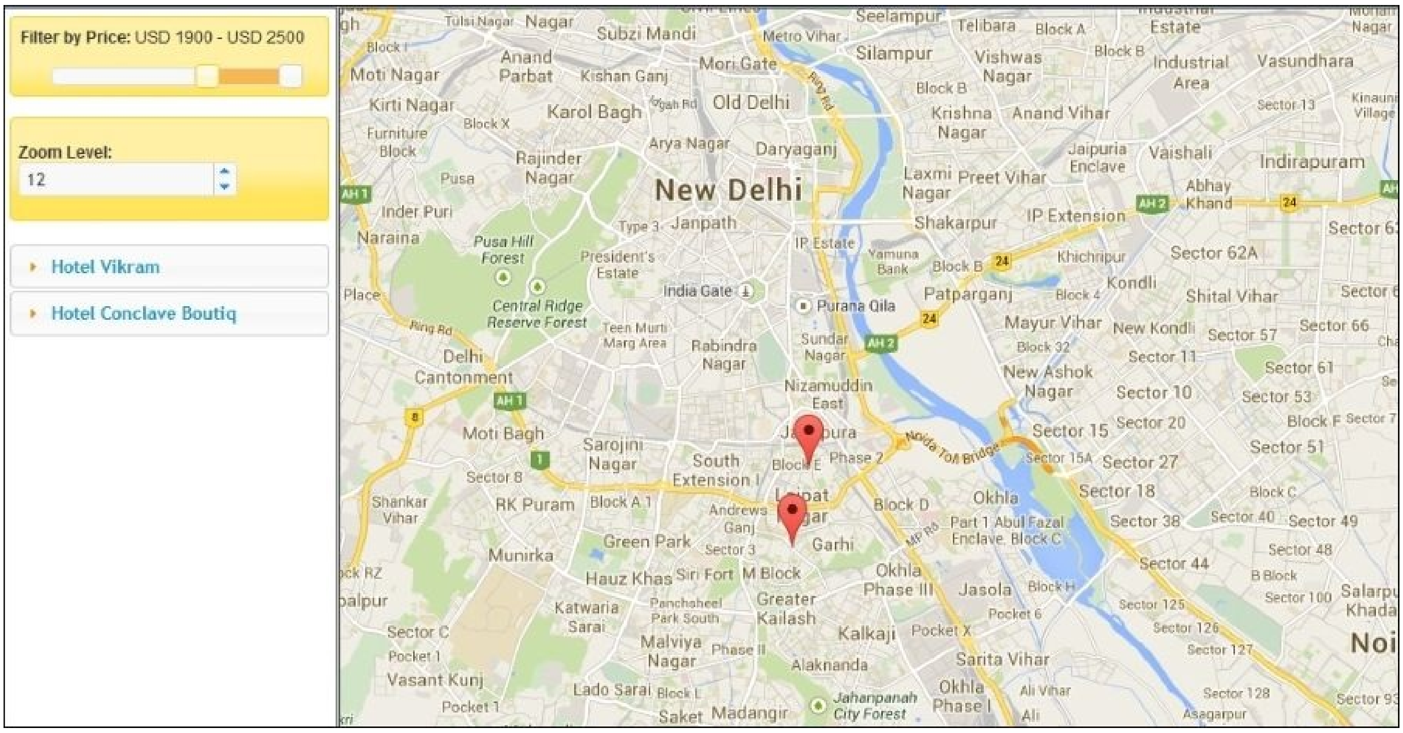
```
setSlider : function()
{
  $('#slider').slider(
  {
    min: myMap.minPrice,
    max: myMap.maxPrice,
    range : true,
    values : [myMap.minPrice, myMap.maxPrice],
    step : 100,
    slide : function(event, ui)
    {
      $('#currentRange').text('USD '+ ui.values[0] + ' - ' + 'USD ' +
ui.values[1]);
    },
    stop : function(event, ui)
    {
      $('#listing h3').each(function()
      {
        var price = parseInt($(this).data('price'), 10);
        //headerIndex corresponds to 0 based index of hotels in object as
well as in DOM
        varheaderIndex = $('#listing h3').index($(this));
        if(price >= ui.values[0] && price <= ui.values[1])
        {
          $('#listing h3:eq('+headerIndex+')').show();
          myMap.markers[headerIndex].setMap(myMap.map);
        }
        else
        {
          $('#listing h3:eq('+headerIndex+')').hide();
          $('#listing div.ui-accordion-
content:eq('+headerIndex+')').hide();
          myMap.markers[headerIndex].setMap(null);
        }
      });
    }
  });
}
```

Since we will allow the user to set the minimum and maximum prices, we have set up a range slider. Let's discuss in detail the properties and methods we have used:

- `min`: This is the minimum value to which a slider would slide.
- `max`: This is the maximum value to which a slider would slide.

- `range`: This is set to true because we want the user to select between `min` and `max` values.
- `values`: In the case of range sliders, we have to provide an array of values where the slider's handles will rest by default. Hence, we have provided `minPrice` and `maxPrice`, which means that the current range is between `minPrice` and `maxPrice`.
- `step`: This is the measure by which the slider will increment in each slide. We have set it to `100`, but you can set it to any other value.
- `slide`: This method is invoked when the user drags the slider. While the user is sliding, we are updating the `currentRange span` element with the slider's currently selected values.
- `stop`: This method is called when the user stops dragging the slider. In this method, we are taking all the `h3` elements inside `div` with `id listing` and iterating over them. The total number of `h3` elements is equal to the number of accordion panels. For each `h3`, we are getting the `price` data attribute, which we defined earlier. Next, we find the index of the current `h3` element among all the `h3` elements. The next `if` block checks whether the price of the current item falls within the range selected by the slider. If it is within the selected range, we show the `h3` element. The next line shows the corresponding marker from the `myMap.markers` array. We have used Google maps' `setMap` method on a marker object to show it on map. If the price is out of the range, we hide the `h3` element and its next `div`, which is the accordion content panel. Along with it, we set the `marker` to `null` as well using the `setMap` again, which removes it from the map.

This makes the last bit of the slider functional as well, and we are ready to check it out. You will see that the slider is visible now, and dragging any of its handles shows or hides respective hotels and markers related to those hotels:



Filter by Price: USD 1900 - USD 2500

Zoom Level: 12

Hotel Vikram

Hotel Conclave Boutiq

Improving the functionality

This application has a lot of scope for enhancements. Here are a few tips to get you started:

- Use hotel ratings with the slider component. Allow the user to filter content based on the rating of the hotel, such as four-star, five-star, and so on.
- Load data of hotels from the backend.
- In the information windows, show real information in the tabs instead of hardcoded information.

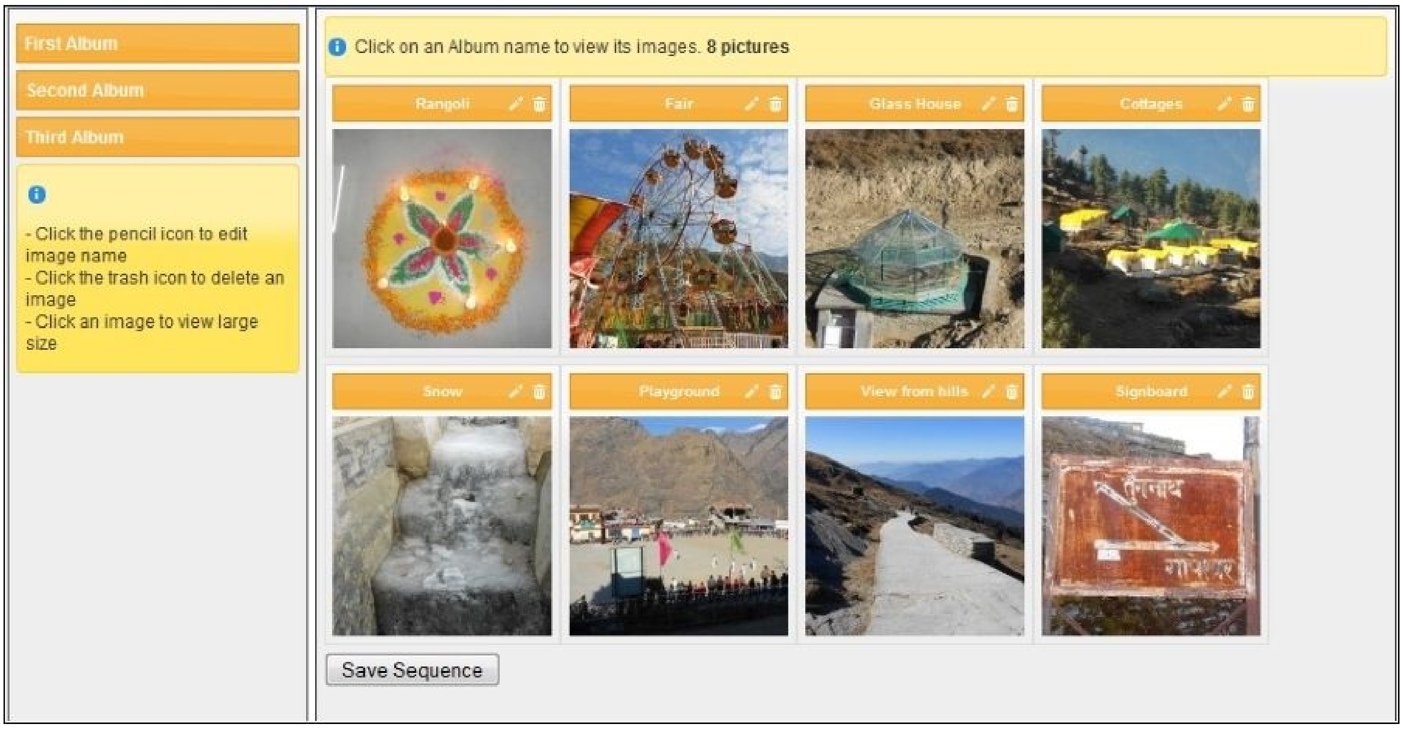
Summary

In this chapter, we covered different components of jQuery UI and the Google Maps API, and saw that mashups can be built easily using different APIs with jQuery UI. In the next chapter, we will create a photo album manager with the help of jQuery UI's sortable and dialog components.

Chapter 8. Creating a Photo Album Manager

So far we have experimented with the jQuery UI components in many new ways and have touched almost all of the components. In Chapter 7, *Using jQuery UI with Google Maps API*, we created a mashup with the help of slider, accordion, tabs, and spinner along with the Google Maps API.

This chapter will teach you to create a simple photo album manager. Users will be presented a list of photo albums on the left-hand side of the page. Clicking any album will display all the pictures in that album on the right-hand side panel. The following is a screenshot of the finished example. Icons for edit and delete are visible towards the right-hand-side of each image caption:

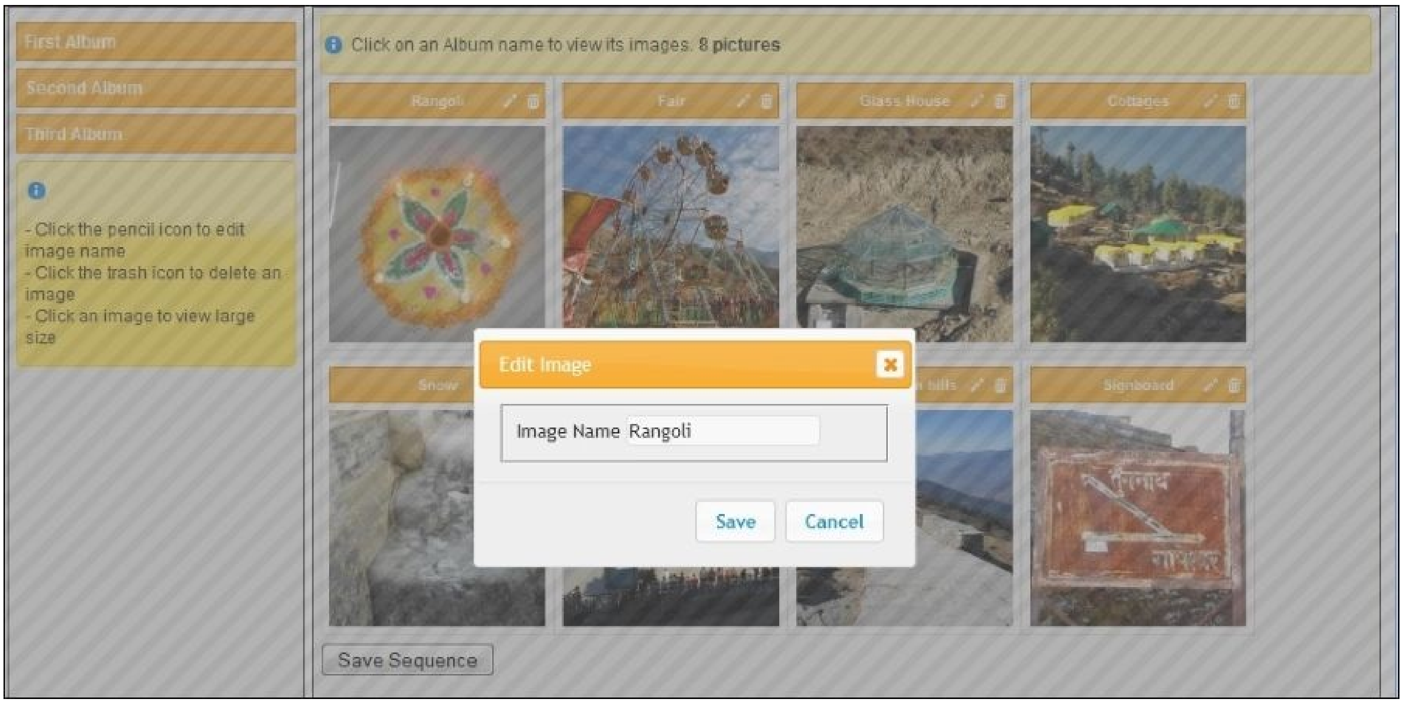


Click on an Album name to view its images. 8 pictures

- Click the pencil icon to edit image name
- Click the trash icon to delete an image
- Click an image to view large size

We will allow users to edit the title of an image and delete an image by providing icons for these actions. Clicking on an image will open its larger version in a dialog box. Users will also be able to rearrange the pictures of an album in a sequence as per their preference.

The following image displays the page that appears when a user clicks the edit icon:


First Album
Second Album
Third Album
- Click the pencil icon to edit image name
- Click the trash icon to delete an image
- Click an image to view large size
Click on an Album name to view its images. 8 pictures
Rangoli
Fair
Glass House
Cottages
Snow
hills
Signboard
Edit Image
Image Name Rangoli
Save
Cancel
Save Sequence

Since our focus is primarily on JavaScript/jQuery and not on backend, we will use a JSON file instead of a database to store albums and picture data. Although you will, most likely, use a database or an API in a practical real-world application, to perform the edit and delete operations and reordering of pictures in an album, we will use a simple server-side script written in PHP. Do not frown if you are unfamiliar with PHP. There will be simple code and you will be able to replace PHP with any server-side language of your choice.

In creating the example in this chapter, we will use the sortable and dialog components of jQuery UI. You will also learn about basic themeroller classes that assist in giving a uniform look to the jQuery UI components.

Creating the folder structure

As usual, we will begin by setting up the folder structure:

1. Create a folder named `Chapter8` inside the `MasteringjQueryUI` folder. Directly inside this folder create an html file and name it `index.html`. This file will keep our html markup.
2. For this chapter, we will need two more files here. Create two files and name them `albums.json` and `ajaxAlbum.php`. The `albums.json` file will keep the albums and pictures data in the JSON format and `ajaxAlbum.php` is the backend script that will be called via AJAX to edit the image name, delete an image, and reorder images of an album.
3. Also, copy the `js` and `css` folders of jQuery UI inside the `Chapter8` folder.
4. Create a new folder named `images` inside the `Chapter8` folder and put eight different images of size 400 px x 400 px inside it.
5. Now create thumbnails of these images of the size 150 px x 150 px. Create another subfolder named `thumbs` inside `images` and put these thumbnails there.
6. Now open the `js` folder and create a new file named `photoAlbum.js`. This file will have all the JavaScript code for our timeline.

That is all we need for the folder structure. Let's now begin by writing the HTML code to make the page.

Designing the page

For the basic markup we will split the page in two parts. The left-hand side will have placeholders for album names and some help text for users. The right-hand side will have a placeholder div to display album pictures.

After these two partitions, there will be markup for dialog boxes to edit, delete, and zoom in to an image.

Creating placeholders for albums and pictures

Since the album names and pictures will be displayed in the page using JavaScript, we only need to specify placeholder divs for these. The following HTML markup for the `index.html` file will be used to divide the page in two parts and create the required elements:

```
<html>
  <head>
    <meta charset="utf-8">
    <title>Photo Album Manager</title>
    <link rel="stylesheet" href="css/ui-lightness/jquery-ui-
1.10.4.custom.min.css">
  </head>
  <body>
    <div class="ui-widget">
      <div class="left ui-widget-content">
        <div id="albumNames"></div>

        <div class="ui-state-highlight ui-corner-all" style="padding:0
5px;">
          <p><span class="ui-icon ui-icon-info" style="float: left; margin-
right: .3em;"></span>&nbsp;
          </p>
          <p>
             - Click the pencil icon to edit image name
            <br>
             - Click the trash icon to delete an image
            <br>
             - Click an image to view large size
          </p>
        </div>

      </div>

      <div class="right ui-widget-content">

        <div class="ui-state-highlight ui-corner-all">
          <p><span class="ui-icon ui-icon-info" style="float: left; margin-
right: .3em;"></span>
          Click on an Album name to view its images.
          <strong id="numImages"></strong>
        </div>

        <ul id="albumPics" class="ui-helper-reset ui-helper-clearfix"></ul>

        <button id="btnSave">Save Sequence</button>

      </div>
    </div>

    <script src="js/jquery-1.10.2.js"></script>
    <script src="js/jquery-ui-1.10.4.custom.min.js"></script>
    <script src="js/photoAlbum.js"></script>
  </body>
</html>
```

We have started by providing a title and including the jQuery UI theme file in the `head` section. In the `body`, we have a `div` with the `ui-widget` class. The `ui-widget` class is from the jQuery UI theme framework that applies font settings for the div and elements inside it. Inside this `div` are two more `div` elements with classes `left` and `right` respectively. Both these divs have another CSS class `ui-widget-content` attached to them. The class `ui-widget-content` is also from the jQuery UI theme framework that sets up styles like the background border and color.

In `div` element having the `.left` class, there is a `div` with an `id` value of `albumNames`. The names of all the albums will be displayed in this container. After this `div` is a highlight box in which we have an `info` icon and some text to help the user.

In `div` element `.right`, we have a highlight div with some text written in it. We will also display the number of pictures in an album in the strong element that has been given the `id` as `numImages`. Next is a `div` with the `id` set as `albumPics`. This is a placeholder unordered list for album images. Clicking an album name from the left panel will display its pictures inside this `ul`. Each picture will be placed inside each individual `li` element. The markup for each `li` will be created from the JSON file. The markup structure will be as follows:

```
<li id="picture_1" class="ui-widget-content">
  <h5 class="ui-widget-header">
    <span id="pictureName_1">Rangoli</span>
    <div class="icons">
      <a data-name="Rangoli" data-id="1" class="ui-icon ui-icon-pencil" 
title="Edit?" href="#"></a>
      <a data-id="1" class="ui-icon ui-icon-trash" title="Delete?" 
href="#"></a>
    </div>
  </h5>
  <a href="images/1.jpg">
    <img width="150" height="150" class="large" src="images/thumbs/1.jpg">
  </a>
</li>
```

Each `li` will have an id and a `ui-widget-content` class. Inside it will be an `h5` element with the `ui-widget-header` class. This `h5` element will have a span element to display the picture name and a div with class icons to display the edit and delete icons. After `h5`, there is an anchor element `<a>` and an image element inside it. The `href` attribute of the anchor will be the path to large image sizes and the `img` element will have its `src` set to thumbnail.

Note

The `id` of the `li` element has been created by appending the picture ID to the `picture_` string.

Note that both the anchor elements inside div having the class `.icons` have data attribute `data-id`.This is the ID of the picture. First anchor has another data attribute `data-name`.

The last element in `div` having `.right` class is a `button` element with `id` `btnSave`. It will be used to save the sequence of pictures in an album.

Finally, before closing the `body` tag, we have included the jQuery and jQuery UI files and most importantly, the `photoAlbum.js` file.

Writing markup for dialog boxes

We will also require three dialog boxes as follows:

- Clicking on the edit icon for a picture will open a dialog box with a text box and **Save** and **Cancel** buttons
- Clicking on the delete icon will open a delete confirmation dialog box with **Delete** and **Cancel** buttons
- Clicking on the thumbnail of a picture will open another dialog box, which will display the large version of that image

For the edit and delete dialogs, we will not create the buttons in markup; instead, they will be created when dialogs will be initialized using jQuery UI. The remaining markup for these dialogs is mentioned here. Place them outside the outer `div` with the `ui-widget` class:

```
<div id="dialogEdit" title="Edit Image" class="dialogBoxes">
  <fieldset>
  <label for="name">Image Name</label>
  <input type="text" id="txtImageName" value="" class="text ui-widget-
content ui-corner-all"/>
  </fieldset>
</div>

<div id="dialogDelete" title="Confirm Delete" class="dialogBoxes">
  <p>
  <span class="ui-icon ui-icon-trash" style="float:left; margin:0 7px 50px
0;"></span>
  Are you sure you want to delete this image?
  </p>
  <p>
    This action is permanent.
  </p>
</div>

<div id="dialogZoom" title="zoom" class="dialogBoxes">
</div>
```

The first dialog has an `id dialogEdit`, and we created a label and a textbox inside it. The textbox will be used by user to input new names for an image.

The second dialog has the id `dialogDelete` and it has a confirmation message inside it.

The last is the dialog with id `dialogZoom`. We will load a large version of an image inside it.

The title attribute of each of these `divs` will become the title of the dialog. All three divs also has a class `dialogBoxes` attached to them which, as we will see in the next section, will hide them.

This is the entire markup we need initially. The rest will be created using jQuery UI after different elements are initialized.

Styling the content

Some basic styles will be required to partition the page into two sections and layout of the pictures in an album. We will need these styles inside the head section to structure the page:

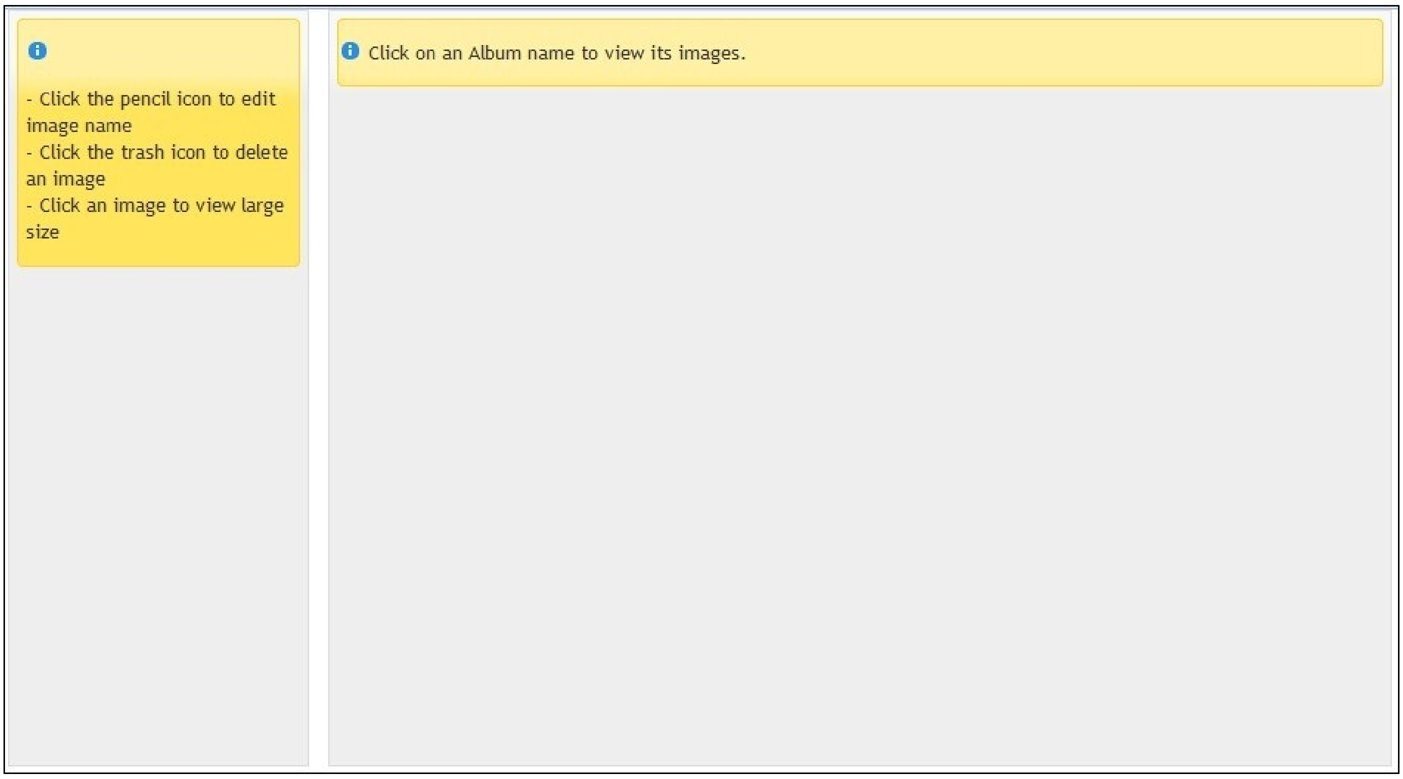
```
<style type="text/css">
  body{
    font-size:12px;
    margin: 0px auto;
    width: 75%;
  }
  .left{
    height:500px;
    width:20%;
    float:left;
    padding:5px;
    margin-left:10px;
  }

  .ui-widget-header{
    margin:5px 0;
    padding:5px;
    cursor:pointer;
  }
  .right{
    height:500px;
    width:75%;
    float:right;
    margin-right:10px;
    position:relative;
    padding:5px;
  }

  #albumPics li{
    float: left;
    height: 190px;
    margin: 0 0 5px;
    padding: 0 5px;
    width: 150px;
    text-align:center;
  }
  #albumPics li h5{
    position:relative;
    cursor:move;
  }
  #albumPics li div.icons{
    position: absolute;
    right: 0;
    top: 3px;
  }

  #albumPics li div.icons a{
    float:left;
  }
  #btnSave{
    display:none;
  }

  .dialogBoxes
  {
    display:none;
  }
</style>
```

We have set up the `.left` class of `div` element with a `width` of `20%` and `height` of `500px`. Similarly, the `.right` class of `div` element has `75% width` and `500px height`. The `div.left` and `div.right` have also been floated to the `left` and `right` respectively.

Next, we provided CSS properties for the `li` elements of the `#albumPics` list and the elements inside it. Each `li` element is 150 px x 190 px in dimension and is floated to the `left`. We have also added some padding to each `li`.

Last are the styles for the `button #btnSave` and `div` having the `.dialogBoxes` class. The `display` has been set to `none` for both of these so that they are not visible when the page loads.

We can now have the first look at our page. Save the `index.html` file and load it on your browser. The page structure will be similar to the following screenshot:



- Click the pencil icon to edit image name
- Click the trash icon to delete an image
- Click an image to view large size

Click on an Album name to view its images.

Not much is visible here but it will change once we start writing some jQuery UI code to display albums.

Creating the JSON file for albums

Before writing any JavaScript code, we need to prepare a JSON file with information about albums and the pictures in it. For this, we will create an array of objects. Each object will represent an album. For this chapter, we will create the JSON with three albums. The first album will have eight images, second will have four, and third will have none. While experimenting, you can of course change the number of images as you wish.

To begin with, let's write the complete JSON structure we will use for this chapter. Open the `albums.json` file in your text editor and write this data into it:

```
[
  {
    "id":"A1",
    "albumName":"First Album",
    "pictures":
    [
      {
        "id":1,
        "sequence":1,
        "imageTitle":"Rangoli",
        "imageThumb":"images/thumbs/1.jpg",
        "imageLarge":"images/1.jpg"
      },
      {
        "id":2,
        "sequence":2,
        "imageTitle":"Fair",
        "imageThumb":"images/thumbs/2.jpg",
        "imageLarge":"images/2.jpg"
      },
      {
        "id":3,
        "sequence":3,
        "imageTitle":"Glass House",
        "imageThumb":"images/thumbs/3.jpg",
        "imageLarge":"images/3.jpg"
      },
      {
        "id":4,
        "sequence":4,
        "imageTitle":"Cottages",
        "imageThumb":"images/thumbs/4.jpg",
        "imageLarge":"images/4.jpg"
      },
      {
        "id":5,
        "sequence":5,
        "imageTitle":"Snow",
        "imageThumb":"images/thumbs/5.jpg",
        "imageLarge":"images/5.jpg"
      },
      {
        "id":6,
        "sequence":6,
        "imageTitle":"Playground",
        "imageThumb":"images/thumbs/6.jpg",
        "imageLarge":"images/6.jpg"
      },
      {
        "id":7,
        "sequence":7,
        "imageTitle":"View from hills",
        "imageThumb":"images/thumbs/7.jpg",
        "imageLarge":"images/7.jpg"
      },
      {
        "id":8,
        "sequence":8,
        "imageTitle":"Signboard",
        "imageThumb":"images/thumbs/8.jpg",
        "imageLarge":"images/8.jpg"
      }
    ]
  },
  {
    "id":"A2",
    "albumName":"Second Album",
    "pictures":
    [
      {
        "id":1,
        "sequence":1,
        "imageTitle":"Snow",
        "imageThumb":"images/thumbs/5.jpg",
        "imageLarge":"images/5.jpg"
      },
      {
        "id":2,
        "sequence":2,
        "imageTitle":"Playground",
        "imageThumb":"images/thumbs/6.jpg",
        "imageLarge":"images/6.jpg"
      },
      {
        "id":3,
        "sequence":3,
        "imageTitle":"Hills",
        "imageThumb":"images/thumbs/7.jpg",
        "imageLarge":"images/7.jpg"
      },
      {
        "id":4,
        "sequence":4,
        "imageTitle":"Sign board",
        "imageThumb":"images/thumbs/8.jpg",
        "imageLarge":"images/8.jpg"
      }
    ]
  },
  {
    "id":"A3",
    "albumName":"Third Album",
    "pictures":[]
  }
]
```

As you can see, there are 3 elements, that is, albums, in this array. Each album has 3 properties, which are `id`, `albumName`, and `pictures`, where `id` refers to the unique id of an album, `albumName` is the display name for that album, and `pictures` is an array again that will have information of all the pictures in that album.

Note

Change the values of the `imageThumb` and `imageLarge` variables to paths of the pictures on your system.

Each element of the `pictures` array represents one picture and it has 5 properties that are described here:

- `id`: This is the unique id of a picture in that album
- `sequence`: This displays the sequence of the image
- `imageTitle`: This is the name of the image
- `imageThumb`: This is the path to the 150 px x 150 px thumbnail of an image
- `imageLarge`: This is the path to the 400 px x 400 px larger version of the same image

With the JSON also ready, we can now proceed to create and display albums. I know you have been waiting so far to write some jQuery UI code. Let's dive in right now.

Getting code structure ready

We will begin by identifying and declaring all the methods required for all the operations. We will first have to retrieve the JSON from the server. Then we will display album names in the left-hand-side panel. After this, we will attach several event handlers for operations such as displaying the pictures of an album, edit image, delete, and so on.

Start with the following code structure in your `js/photoAlbum.js` file, which defines an object literal to wrap the functionality for the album manager and a document ready handler for jQuery:

```
var albums =
{
  jsonAlbums : null,
  currentAlbum : null,
  currentPictureId : null,
  initialize : function()
  {
    $.getJSON( "albums.json", function(data)
    {
      albums.jsonAlbums = data;
      albums.fillAlbumNames();
      albums.addEventHandlers();
    });
  },
  fillAlbumNames : function()
  {
  },
  addEventHandlers : function()
  {

  },
  displayAlbum : function(albumId)
  {
  },
  editImage : function()
  {
  },
  deleteImage : function()
  {
  },
  saveNewSequence : function()
  {
  }
};

$(document).ready(function()
{
  albums.initialize();
});
```

The album's object contains all the properties and methods we need to create an album manager. Let's have a look at each of its properties:

- `jsonAlbums`: After retrieving the JSON from the server, we will store it in this variable
- `currentAlbum`: This is the `id` of the current album being viewed. It will be set when an album name is clicked on from the panel on the left-hand side.
- `currentPictureId`: This is the `id` of an individual picture. We will need it for `edit` and `delete` operations.
- `initialize`: This will be the starting functions that will make the AJAX call to server and start the process to build the page.
- `fillAlbumNames`: As the name suggests, this method will display the names of albums in the left-hand side panel.
- `addEventHandlers`: Event handlers to handle all the events on the page will be added inside this method.
- `displayAlbum`: This method will display the pictures of an album. It will be called when an album name on the left panel is clicked. The `id` of the album will be passed to it.
- `editImage`: This method will save the edited name for an image in the source JSON file
- `deleteImage`: This method will delete an image from the source JSON file
- `saveNewSequence`: This will be called when the **Save Sequence** button is clicked on. It will save the new order of images in an album after the user has rearranged the sequence by sorting the images. This will change the sequence in the source JSON file

After this object is the `$(document).ready()` handler for jQuery, which simply calls the `initialize` method of the album's object. Therefore, we will start by implementing the initialize method first.

Implementing the initialize method

In this method, we will load the JSON from the server and display the album names in the left panel. Write the following code inside the initialize method to get the JSON from the server:

```
$.getJSON( "albums.json", function(data)
{
  albums.jsonAlbums = data;
  albums.fillAlbumNames();
  albums.addEventHandlers();
});
```

Here we are using jQuery's `getJSON` method to fetch JSON from the server. We have specified `albums.json` as the URL and a success handler when a response is received. In success event handler, response is received in a variable `data` that we place in property `jsonAlbums`.

Next, we call the methods `fillAlbumNames` and `addEventHandlers` of the `albums` object.

Filling album names

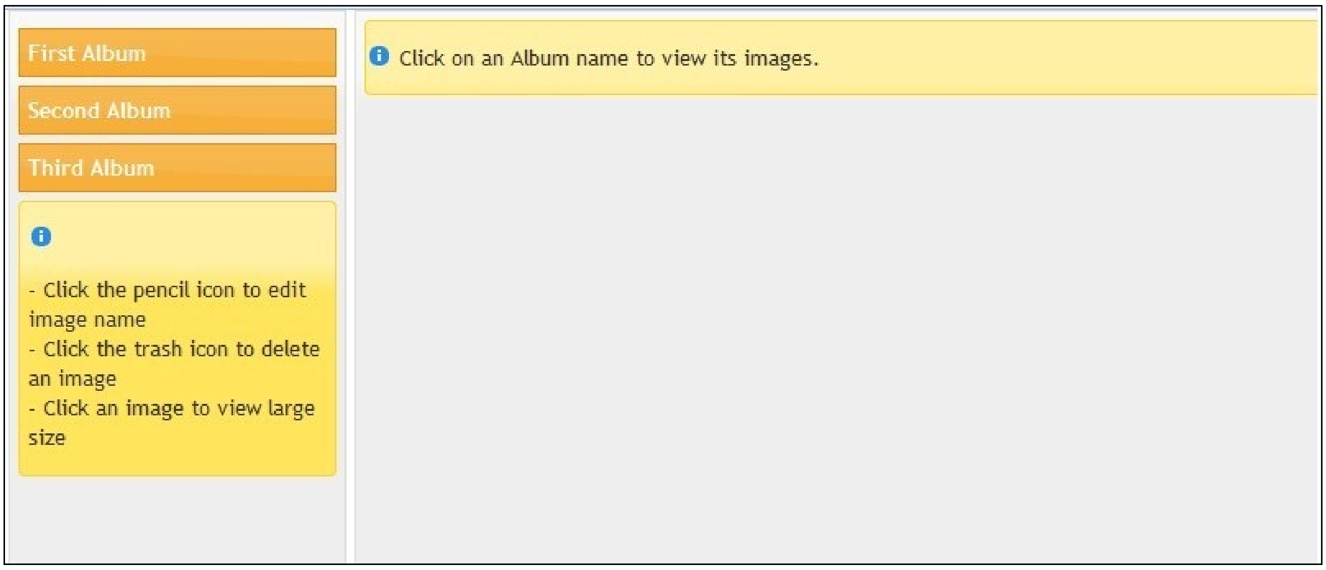
To fill the album names, we will iterate in the response JSON and create HTML with album names. We will then place this HTML inside the placeholder in the left panel:

1. Write this code for method `fillAlbumNames` to display albums in the left panel:

   ```
   var albumNames = [];
   $.each(this.jsonAlbums, function(key, album)
   {
     albumNames.push('<h4 class="ui-widget-header album" data-id="' +
   album.id + '">' + album.albumName + ' </h4>');
   });
   $('#albumNames').html(albumNames.join(''));
   ```

   We have declared an array `albumNames` that will hold the DOM structure for each album.

2. Next we iterate in `jsonAlbums` property using jQuery's `$.each` iterator. In each iteration we create an `h4` element with the album name inside it.
3. We also attach CSS classes `ui-widget-header` and `album` to it, and a data attribute `data-id`, which is set to the `id` of the `album`, has also been added.
4. After `$.each` is finished, we push the DOM inside the div `albumNames`.
5. You can now verify that album names are being displayed in the left panel if you reload the `index.html` page:



First Album
Second Album
Third Album

- Click the pencil icon to edit image name
- Click the trash icon to delete an image
- Click an image to view large size

Click on an Album name to view its images.

Displaying the albums

Once album names are visible in the left panel, our first task now is to add event handlers. The first event handler that we will add will be to display pictures when an album name is clicked on. Then we will proceed gradually towards other event handlers.

Since each album name has a class name `album` attached to it, we can add an event handler for the `click` event of this class name. Go to the `addEventHandlers` method in albums object literal and write the following code for the event handler:

```
$('.album').on('click', function()
{
  albums.displayAlbum($(this).data('id'));
});
```

The `click` handler calls the method `displayAlbum`. The argument passed to `displayAlbum` is the `id` of the album. Note that we had provided a `data-id` attribute for each album in the previous section.

Let's now define the `displayAlbum` method. In this method, we will iterate in the `jsonAlbums` object and create HTML to display all the pictures in the album. Go to the `displayAlbum` method and write the following code to create DOM and push it into the page:

```
$('#albumPics').empty();
$('#btnSave').hide();
this.currentAlbum = albumId;
var listItems = '';
for(var i = 0; i < this.jsonAlbums.length; i++)
{
  if(this.jsonAlbums[i].id == albumId)
  {
    if(this.jsonAlbums[i].pictures.length > 0)
    {
      var allPictures = this.jsonAlbums[i].pictures;
      /* sort pictures by sequence before displaying*/
      allPictures.sort(function(a,b)
      {
        return a.sequence - b.sequence;
      });
      $.each(allPictures, function(key, picture)
      {
        listItems+= '<li class="ui-widget-content" id="picture_'+
picture.id +'">';
          listItems+= '<h5 class="ui-widget-header"><span
id="pictureName_'+  picture.id +'">'+ picture.imageTitle + '</span>';
            listItems+= '<div class="icons">';
              listItems+= '<a href="#" title="Edit?" class="ui-icon ui-
icon-pencil" data-id="' + picture.id +'" data-name="' + picture.imageTitle
+ '"></a> ';
              listItems+= '<a href="#" title="Delete?" class="ui-icon ui-
icon-trash" data-id="' + picture.id +'"></a>';
            listItems+= '</div>';
          listItems+= '</h5>';
          listItems+= '<a href="' + picture.imageLarge + '">';
            listItems+= '<img src="' + picture.imageThumb + '" width="150"
height="150" class="large">';
          listItems+= '</a>';
        listItems+= '</li>';
      });
      $('#btnSave').show();
    }
    else
    {
      listItems+= '<li class="ui-widget-content">No pictures in this
album</li>';
    }
    $('#numImages').text(this.jsonAlbums[i].pictures.length + ' pictures');
    $('#albumPics').html(listItems);
    break;
  }
}
```

We start by emptying the `#albumPics` list and hiding the **Save** button. Then we set the `currentAlbum` property to `albumId` and create a variable named `listItems`, which will hold the DOM for album pictures.

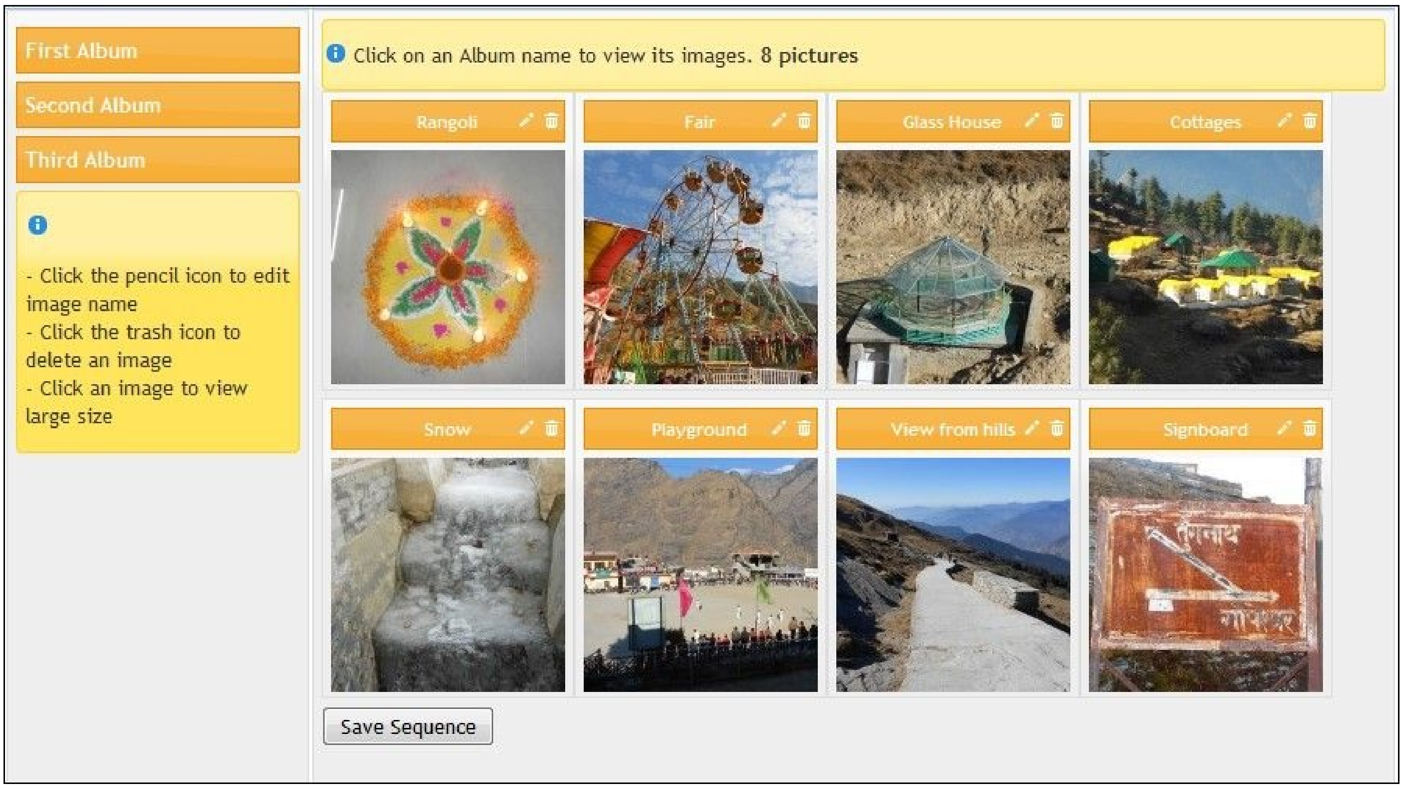
Next we iterate over `jsonAlbums` and compare the `id` of each album in JSON to the variable `albumId`. When a match is found, we check whether there are any pictures available for this album. In case the `pictures` array of an album has more than `0` items, we store them in a variable `allPictures`. Then we sort the pictures array by key `sequence`.

After this, we iterate over all the pictures using `$.each` and create DOM by appending the HTML string to the `listItems` variable in each iteration. We have already discussed the structure of `li` elements in the *Creating placeholders for albums and pictures* section.

Once iteration is complete, we display the **Save** button. If there are no pictures in the album, we create an appropriate message for DOM.

After the `if-else` condition, we display the number of pictures in an element with `id` `numImages`. And finally, we push the newly created HTML that is in the variable `listItems` inside list `#albumPics`.

Save the `photoAlbum.js` file and hit reload for `index.html` in your browser. Click on any album name in the left panel and you will see beautiful pictures in the album in the grid format:



Click on an Album name to view its images. 8 pictures

- Click the pencil icon to edit image name
- Click the trash icon to delete an image
- Click an image to view large size

Making the pictures sortable

To make the pictures sortable, go to the `addEventHandlers` method again and append this code to initialize the sortable component of jQuery UI:

```
$('#albumPics').sortable(
{
  handle : '.ui-widget-header',
  placeholder: "ui-state-highlight",
  cursor:'move'
});
```

We have initialized the sortable components with three options:

- `handle`: Normally, sortable elements can be moved by dragging any part of the container. The `handle` option allows us to provide a custom handle, which could be only a part of a sortable element, to move the element. In our case, each list item that displays the picture is made up of two parts; the first is an `h5` element that displays image name and icons and acts as a header. The second is the anchor that holds the image. Since we want to reserve the anchor for zooming the image, we have made the `h5` element the handle. Since the `h5` element has the `ui-widget-header` class attached to it, we have provided this class name for the `handle` option.
- `placeholder`: When a sortable element is moved, its original position becomes empty. By providing a placeholder value, a class is applied to the empty space, which we can customize. Here we have provided `ui-state-highlight` as the placeholder value. This class name is from jQuery UI the framework that applies a highlight effect.
- `cursor`: This is the style for the mouse pointer when a sortable element is moved. We have set it as `move`.

After initializing the sortable, you can check it on the browser. You will be able to rearrange the pictures by moving them using the specified handle.

Initializing dialogs for edit, delete, and zoom

We have also written markup for three dialogs that are used for editing an image name, deleting an image, and zooming an image. In order to use those dialogs, we will first have to initialize them and convert them into jQuery UI dialog components.

The code for initializing these dialogs will be written inside the `addEventHandlers` method as well. Go ahead and write this code after the sortable initialization to initialize the dialogs:

```
$("#dialogEdit").dialog(
{
  resizable: false,
  autoOpen : false,
  modal: true,
  buttons:{
    Save: function()
    {
      albums.editImage();
    },
    Cancel: function()
    {
      $('#txtImageName').val('');
      albums.currentPictureId = null;
      $(this).dialog( "close" );
    }
  }
});

$("#dialogDelete").dialog(
{
  resizable: false,
  autoOpen : false,
  modal: true,
  buttons:
  {
    Delete: function()
    {
      albums.deleteImage();
    },
    Cancel : function()
    {
      albums.currentPictureId = null;
      $(this).dialog( "close" );
    }
  }
});

$("#dialogZoom").dialog(
{
  resizable: false,
  autoOpen : false,
  modal: true,
  position : "top",
  width:430,
  show : 'scale',
  hide : 'scale'
});
```

Three options are common while creating all three dialogs. These properties are `resizable`, `autoOpen`, and `modal`. Setting the `resizable` option to `true` will allow the user to resize the dialog box by dragging it from its bottom right corner. We have disabled this in our code. The `autoOpen` property defines whether a dialog will open automatically upon initialization or not. We have set it to `false` because we want to open it manually:

- For `dialogEdit` we have defined two buttons, **Save** and **Cancel**:
  - Clicking on the **Save** button will call the `editImage` method, which we will implement later
  - Clicking on **Cancel** will empty the textbox `txtImageName`, set `currentPictureId` to `null`, and close the dialog
- For `deleteDialog`, the two buttons are **Delete** and **Cancel**:
  - Clicking on **Delete** will call the `deleteImage` method, which we will implement later
  - Clicking on **Cancel** will set the `currentPictureId` to `null` and close the dialog
- The last dialog is `dialogZoom`:
  - For this dialog, we have defined `position` as `top` so that it appears at the top of viewport
  - Since large image size of a picture is 400 px, we have defined `width` as `430`, adding for padding towards left and right.
- The `show` and `hide` options define which effects will be used when dialog opens and closes. We have set it to the `scale` effect.

The dialogs have been initialized now and we can display them when the user clicks on the edit and delete icons or tries to zoom in to an image. We will handle these events in the following sections.

Handling click events for edit, delete icons, and zooming pictures

Since all the icons are inside `ul#albumPics`, instead of adding events for each item individually, we can add an event handler on `ul#albumPics` and then check which element was clicked on. All event handlers are passed an event object. We can use the target property of this event object to check which element was clicked on and take action accordingly.

Still inside the `addEventHandlers` method, this code will be used to handle edit, delete, and image zoom:

```
$('ul#albumPics').on('click', function(event)
{
  var target = $(event.target);
  if(target.is('a.ui-icon-pencil') )
  {
    var pictureId = target.data('id');
    var pictureName = target.data('name');
    albums.currentPictureId = pictureId;
    $('#txtImageName').val(pictureName);
    $( "#dialogEdit" ).dialog('open');
  }
  else if(target.is('a.ui-icon-trash') )
  {
    var pictureId = target.data('id');
    albums.currentPictureId = pictureId;
    $("#dialogDelete").dialog('open');
  }
  else if(target.is('img.large') )
  {
    var largeImagePath = target.parent().attr('href');
    $('#dialogZoom').html('<img src="' + largeImagePath +
'">').dialog('open');
  }
  return false;
});
```

Inside the event handler, we have taken the target element in a variable named `target`. The jQuery function `is` is then used to check which element was clicked on.

The first `if` condition checks whether the clicked element was the edit icon as the edit icon has the `ui-icon-pencil` class. In this case, we take the `id` and `name` of the picture using the data attributes `data-id` and `data-name` from the edit icon. Then we set the `currentPictureId` property to the `pictureId`.

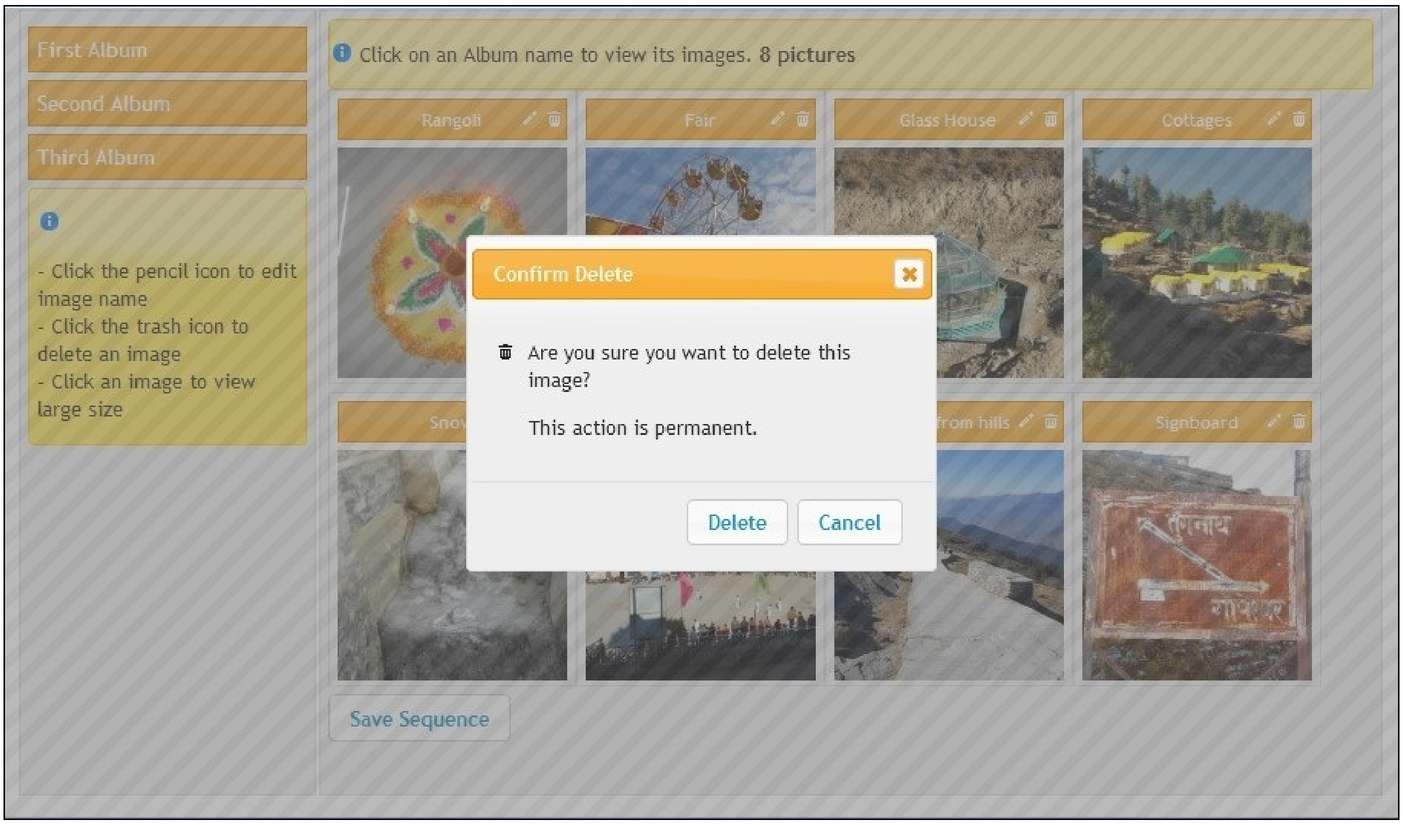
We then fill the picture name in the text box with the id `txtImageName`. Finally, we open the `dialogEdit` dialog.

The second condition checks whether the clicked element was the Delete icon. In this case, we take the picture id from the `data-id` attribute and set the `currentPictureId` property. Then we open the `dialogDelete` dialog.

The last condition checks whether the thumbnail image itself was clicked on. If this is the case, we get the value of the `href` attribute of the image's parent anchor. If you recall, the value of the `href` attribute is the path to the large image. We then create an image element and push it inside `div#dialogZoom` and then call the `open` method of the dialog ,which will display the dialog with the `scale` effect.

As the icons are anchors and the image is also wrapped inside an anchor, `return false` is used to disable the default behavior.

Reload the `index.html` page on the browser and try clicking on the Edit and Delete icons. Respective dialogs will open for these:

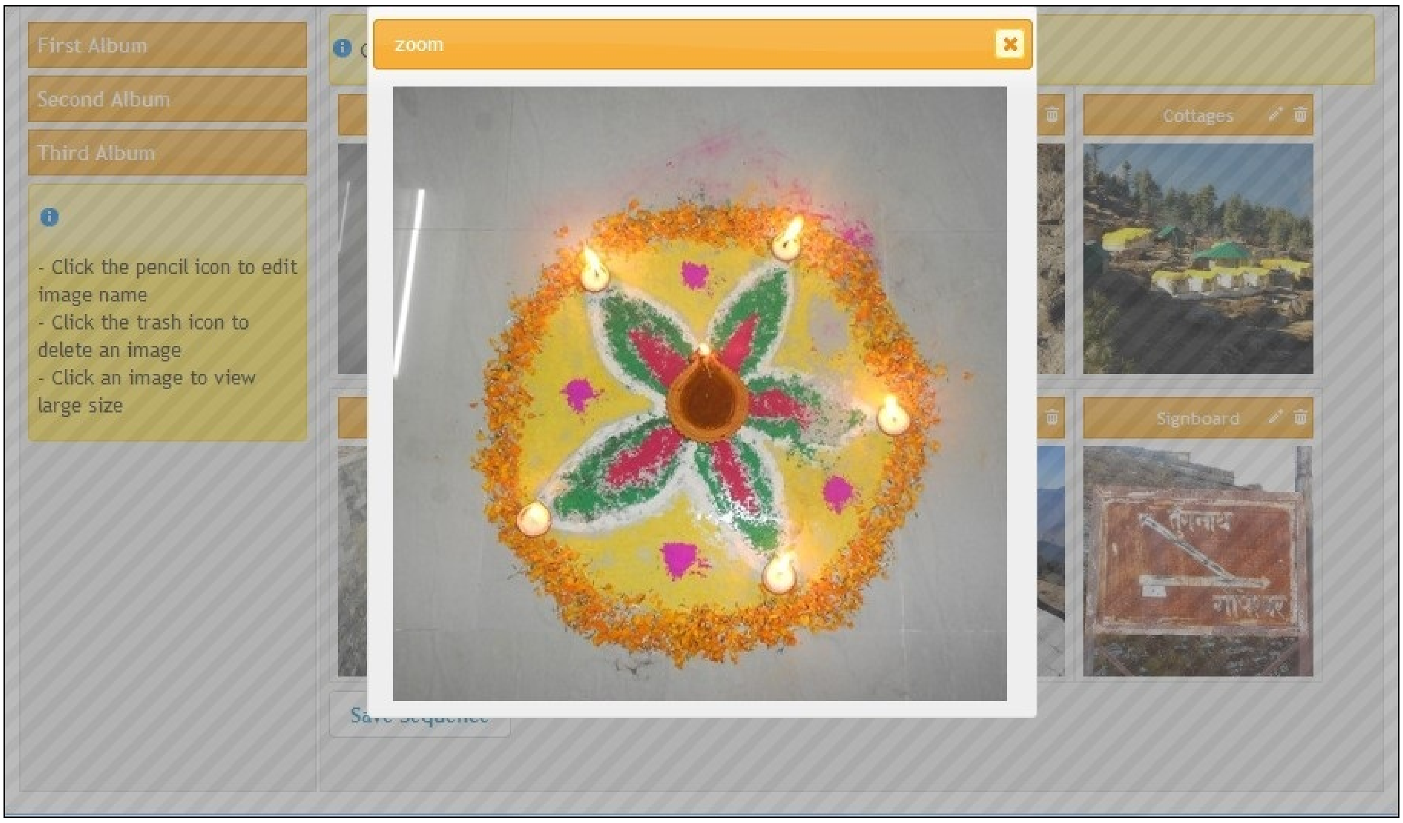

First Album
Second Album
Third Album
- Click the pencil icon to edit image name
- Click the trash icon to delete an image
- Click an image to view large size
Click on an Album name to view its images. 8 pictures
Rangoli
Fair
Glass House
Cottages
Sno
from hills
Signboard
Save Sequence
Confirm Delete
Delete
Cancel

Are you sure you want to delete this image?

This action is permanent.

Nothing will happen on clicking either the **Save** and **Delete** buttons, as we have not yet implemented them.

However, clicking on the thumbnail will open a dialog box and will display a large version of the image in it. You will also see the scale effect when the dialog opens and closes:



- Click the pencil icon to edit image name
- Click the trash icon to delete an image
- Click an image to view large size

Last event handler inside the `addEventHandlers` method is the click handler for `btnSave`. Let's define it as well:

```
$('#btnSave').button().on('click', function()
{
  albums.saveNewSequence();
});
```

Clicking on the **Save Sequence** button will call the `saveNewSequence` button, which we are going to implement in the next section along with the edit and delete operations.

Editing, deleting, and rearranging pictures

Before we code, let’s see how the edit, delete, and rearrange operations will work.

On clicking on the edit icon, a dialog box appears with the image name in a text box. The user will edit this name and click on the **Save** button. Clicking on the **Save** button will send an AJAX request to a PHP script. The album ID, picture ID, and edited name will be sent to this script. Depending on the album ID and picture ID, the PHP script will change the name of said image in the original `albums.json` file.

Similarly, to delete an image, we will send the album id and picture id to the PHP script. The PHP script will delete the image from the original `albums.json` file.

To rearrange the sequence of images, we will get the current sequence on the page, which the user has made after reordering the images. We will send this sequence to the PHP script, and the script will update sequence values for each picture in album.

Editing a picture name

On clicking on the **Save** button of the edit image dialog, the `editImage` method will be called that will send the required AJAX request. Here is how we will define this method:

```
var editImageRequest = $.ajax(
{
  url : 'ajaxAlbum.php',
  type: "POST",
  data: { action : 'edit', albumId: albums.currentAlbum, pictureId: 
albums.currentPictureId, newImageName : $('#txtImageName').val() }
});

editImageRequest.done(function(data)
{
  $.getJSON( "albums.json", function( data )
  {
    albums.jsonAlbums = data;
    $('#pictureName_' + 
albums.currentPictureId).text($('#txtImageName').val());
    $( "#dialogEdit" ).dialog('close');
  });
});

editImageRequest.fail(function( xhr, status )
{
  alert( "Error - " + status );
});
```

In the preceding code, we send a `POST` request to the `ajaxAlbum.php` file. Four parameters are also being sent along with the request. These parameters are as follows:

- `action`: This is required by the server-side script to identify which operation to perform. We have sent `edit` in this case.
- `albumId`: This is the ID of the album whose picture is being edited. We had set it in `albums.currentAlbum` when the album name was clicked on.
- `pictureId`: This is the ID of the picture whose name is being edited. We had set it in `albums.currentPictureId` when the edit icon was clicked on.
- `newImageName`: This is the name of the image specified by the user in the text box.

Success callback for this AJAX request is defined next using the `.done` method. Once the PHP script is done, the control will reach here and any response from the server will be received via a parameter to this method. In our case, nothing is being returned from the server.

Since the original JSON has changed now, we need to update the `jsonAlbums` property, which still has old JSON data. To do this, we make an AJAX call again with `getJSON` and place the response in `jsonAlbums`. Then the dialog is closed.

In case of an error, we have an error callback as well, which will display the error message in the alert box.

Deleting a picture

Similar to edit, an AJAX request will be sent when the user clicks the **Delete** button on the dialog box. Go to the `deleteImage` method and write this code to send an AJAX request to delete the image:

```
var deleteImageRequest = $.ajax(
{
  url : 'ajaxAlbum.php',
  type: "POST",
  data: { action : 'delete', albumId: albums.currentAlbum, pictureId: 
albums.currentPictureId }
});

deleteImageRequest.done(function(data)
{
  $.getJSON( "albums.json", function( data )
  {
    albums.jsonAlbums = data;
    albums.displayAlbum(albums.currentAlbum);
  });
  $( "#dialogDelete" ).dialog('close');
});

deleteImageRequest.fail(function( xhr, status )
{
  alert( "Error - " + status );
});
```

In this AJAX request, we only need to send the album ID and the picture ID we want to delete along with the key action whose value will be delete this time.

Similar to edit image, we have defined a success event handler that will get the updated `albums.json` file from the server and put it in the `albums.jsonAlbums` file.

Since an image has been deleted, we need to refresh the UI as well. Hence, we call the `displayAlbum` method again for the current album.

Refreshing the page and selecting an album will show that deleted images have been removed permanently from the original JSON file.

Rearranging pictures of an album

To rearrange pictures, we need to get the new sequence of images, which is in the page. Clicking on the **Save Sequence** button will send the AJAX request with this sequence and the PHP script will do the rest. This can be done easily using the serialize method of the sortable component. Go to the `saveNewSequence` method and write this code:

```
var x = ($('#albumPics').sortable('serialize'));
var editSequenceRequest = $.ajax(
{
  url : 'ajaxAlbum.php',
  type: "POST",
  data: 'action=reorder&albumId='+ albums.currentAlbum +  '&' + x
});
editSequenceRequest.done(function(data)
{
  $.getJSON( "albums.json", function( data )
  {
    albums.jsonAlbums = data;
    albums.displayAlbum(albums.currentAlbum);
  });
});

editSequenceRequest.fail(function( xhr, status )
{
  alert( "Error - " + status );
});
```

Here's what we did in the code:

- First of all we used the `serialize` method to get new sequence of IDs.
- Note that we have already created IDs of sortable elements as `picture_1`, `picture_2`, `picture_3`, and so on, where the number after underscore(_) is the image ID.
- Using the `serialize` option of the `sortable` component will create a hash that will be something like `picture[]=1& picture[]=2& picture[]=3`. On server side these IDs will be received in an array.
- An AJAX request is then sent with action set to reorder, album ID set to `currentAlbum`, and the hash that is stored in variable `x`.
- The success handler for this request will be similar to the delete operations success handler. We will fetch the updated JSON using `getJSON`, set value of `jsonAlbums`, and call the `displayAlbum` method again.

You can now refresh the page and see that images of albums will appear in the sequence you saved them.

The ajaxAlbum.php file

All the AJAX calls in previous sections were sent to the `ajaxAlbum.php` file. This is the PHP file that will make all changes to the JSON file `albums.json`. We will place a `switch` case in this file to identify paths for edit, delete, and reorder. Since PHP is not in the scope of this chapter or book, we will not go through server-side code in too much detail. The following is the code that will make the changes:

```
<?php
  $albumId = $_POST['albumId'];
  $pictureId = $_POST['pictureId'];

  $jsonAlbums = file_get_contents('albums.json');
  $jsonAlbums = json_decode($jsonAlbums);
  switch($_POST['action'])
  {
    case 'edit':
      foreach($jsonAlbums as $album)
      {
        if($album->id == $albumId)
        {
          foreach($album->pictures as $picture)
          {
            if($picture->id == $pictureId)
            {
              $picture->imageTitle = $_POST['newImageName'];

              file_put_contents('albums.json', json_encode($jsonAlbums));

              break;
            }
          }
          break;
        }
      }
    break;
    case 'delete':
      foreach($jsonAlbums as $album)
      {
        if($album->id == $albumId)
        {
          foreach($album->pictures as $index => $picture)
          {
            if($picture->id == $pictureId)
            {
              unset($album->pictures[$index]);
              $remaining = array_values($album->pictures);
              $album->pictures = $remaining;
              file_put_contents('albums.json', json_encode($jsonAlbums));
              break;
            }
          }
          break;
        }
      }
    break;
    case 'reorder':
      $pictureIds = $_POST['picture'];
      foreach($jsonAlbums as $album)
      {
        if($album->id == $albumId)
        {
          $sequenceStart = 1;
          foreach($pictureIds as $id)
          {
            /* find this id in in album pictures and set sequence*/
            foreach($album->pictures as $picture)
            {
              if($picture->id == $id)
              {
                $picture->sequence = $sequenceStart;
                $sequenceStart++;
                break;
              }
            }
          }
          file_put_contents('albums.json', json_encode($jsonAlbums));
          break;
        }
      }
    break;
  }
?>
```

The first and second lines get the album ID and picture ID in variables `$albumId` and `$pictureId`, respectively. Then we load the `albums.json` file and convert it into an object.

Note

A key named action will be sent with each AJAX request so that the PHP script may identify which operation to perform. The possible values are edit, delete, and reorder for editing, deleting, and rearranging images, respectively.

Depending on the value of the action, any of the three operations take place.

For editing the image, we iterate in the JSON and after finding the correct album and picture based on `$albumId` and `$pictureId`, we update the name. After updating the name, we convert the object to JSON and write it back in the `albums.json` file with the help of the PHP `file_put_contents` function.

Similarly, the delete operation is performed. After finding the correct album and image, the array element from the pictures array is removed. The remaining JSON is written to `albums.json` file, like before.

To reorder the images we have an array of the latest sequence in the form of an array in the variable picture that was sent in the AJAX request. Then for the specified album, we loop over all the ids sent from the browser and set a sequence accordingly. The modified object is converted back to JSON and written in the `albums.json` file.

Note

Make sure that you have write permissions on the `albums.json` file. Without this, no changes will be written to the file.

Improving album manager

There is a lot of scope for improvement in this album manager. You can start with these suggestions:

- Use a database instead of a JSON file
- Use a Flickr API instead of a JSON file
- Allow users to move pictures from one album to another (Hint: use connected sortables)
- Use jQuery UI’s slider component to zoom in and zoom out of an image

Summary

This was a long chapter with a lot of functionalities to cover. You learned to create a basic photo album manager application. We were able to edit image description and delete an image from an album. We were also able to reorder the pictures in an album. In this process, you made use of jQuery UI's sortable and dialog components and learned about some themeroller classes as well.

Going forward, the next chapter is going to be interesting as you will learn to create your own widgets using jQuery UI's widget factory.

Chapter 9. Creating Widgets Using the Widget Factory

So far, we worked with the default components or widgets provided by the jQuery UI library. In this chapter, we will take a step forward and use the power of jQuery UI's widget factory to create our own widgets.

The widget factory of jQuery UI is the basic building block of all the components of jQuery UI. This means that if you create your own widgets using the widget factory as a base, you are doing so in a standardized manner similar to the way other native widgets have been written. This helps in maintaining consistency among all widgets. Another advantage is that the widget factory follows an object-oriented approach to create a widget.

We will use the widget factory in this chapter to create two widgets. Our first widget will be used to search for a string in the rows of a HTML table. Initializing this widget for a table will create an input textbox in the first row of the table. Typing any characters in the textbox will start a search for the matching string in the table. Based on the input string, only matching rows will be visible and any nonmatching rows will be hidden. We will also allow users to customize the options for the widget as well.

The second widget will convert a list of elements into a slideshow. Elements will be displayed one by one after a fixed duration. We will allow users to customize this widget with options, have such as the duration for which an element will be displayed, effects used while showing and hiding an element, and the speed of the "show and hide" effects. We will also provide a callback method that will be available for users of the widget to implement. This callback will be called before any element is displayed.

Both of these widgets will cover the concepts of custom widget creation, and you will be able to create similar and advanced widgets like these easily in future.

The folder structure

The folder structure for this chapter will be same as that of the previous chapters. Create a folder named `Chapter9` inside the `MasteringjQueryUI` folder. Inside the `Chapter9` folder, create two HTML files and name them `search.html` and `slides.html`, respectively. Also, copy the `js` and `css` folders inside the `Chapter9` folder. Now, open the `js` folder and create two more JavaScript files with the names `searchable.js` and `slides.js`. These files will be used to write widget-specific codes for `searchable` and slide widgets, respectively.

With the folder setup complete, let's start to build our first widget: `searchable`.

Creating a widget to search data in a table

The `searchable` widget will insert a new row at the beginning of the `table` element. Inside this row, there will be a `td` element that will have a `label` and a `textbox` for searching.

For the markup on the HTML page, we will only need a table with some rows.

Writing markup for the table

The entire markup will be written inside the `search.html` file. We will create a table with three columns. The first column will be the serial number, the second column will be the name of a place, and the third column will have comma-separated tags related to that place. To create such a table, open the `search.html` file and write the following code to create a regular HTML table:

```
<html>
  <head>
    <title>Searchable Widget</title>
    <link rel="stylesheet" href="css/ui-lightness/jquery-ui-
1.10.4.custom.min.css">
  </head>
  <body>
    <p>
      <button id="btnEnable">Enable Searchable</button>
      <button id="btnDestroy">Destroy Searchable</button>
    </p>
    <table width="100%" id="tblData" class="tables">
      <tbody>
        <tr>
          <th width="10%">#</th>
          <th width="35%">Place</th>
          <th width="55%">Tags</th>
        </tr>
        <tr>
          <td class="odd">1</td>
          <td class="odd">Pithoragarh</td>
          <td class="odd">Hills, Snow</td>
        </tr>
        <tr>
          <td class="even">2</td>
          <td class="even">Dhakuri</td>
          <td class="even">Trekking, Himalayas, Camping</td>
        </tr>
        <tr>
          <td class="odd">3</td>
          <td class="odd">Goa</td>
          <td class="odd">Beach, Fun, Holidays</td>
        </tr>
        <tr>
          <td class="even">4</td>
          <td class="even">Nainital</td>
          <td class="even">Snow, Lake, Hills</td>
        </tr>
        <tr>
          <td class="odd">5</td>
          <td class="odd">Dayara Bugyal</td>
          <td class="odd">Trekking, Camping</td>
        </tr>
        <tr>
          <td class="even">6</td>
          <td class="even">Mumbai</td>
          <td class="even">Beach, Bollywood</td>
        </tr>
        <tr>
          <td class="odd">7</td>
          <td class="odd">Agra</td>
          <td class="odd">Taj Mahal, Holiday</td>
        </tr>
        <tr>
          <td class="even">8</td>
          <td class="even">Ranikhet</td>
          <td class="even">Sunset, Hills</td>
        </tr>
        <tr>
          <td class="odd">9</td>
          <td class="odd">Auli</td>
          <td class="odd">Skiing, Snow, Honeymoon</td>
        </tr>
        <tr>
          <td class="even">10</td>
          <td class="even">Chopta</td>
          <td class="even">Trekking, Honeymoon</td>
        </tr>
      </tbody>
    </table>

    <script src="js/jquery-1.10.2.js"></script>
    <script src="js/jquery-ui-1.10.4.custom.min.js"></script>
    <script src="js/searchable.js"></script>
  </body>
</html>
```

The preceding code begins by referencing the path to the jQuery UI CSS file inside the `head` section. There is not much to do for this file in this example. We are using it just to beautify the buttons. You can avoid including it altogether.

Then, inside the `body` tag, we defined two buttons. These buttons have the `id` `btnEnable` and `btnDestroy` and these will be used to enable and destroy the widget, respectively.

Then, we created the table that has the `id` set to `tblData` and `class` set to `tables`. Inside it, there are 10 rows, each having three columns with some data in each of them. Note that each alternate row has classes `odd` and `even` applied to its cells. This is just for presentation purpose and is not in any way related to the widget.

After creating the table, we referenced the jQuery core file, the jQuery UI core file, and finally a reference to the `searchable.js` file where we will write the code to create the widget.

Styling the content

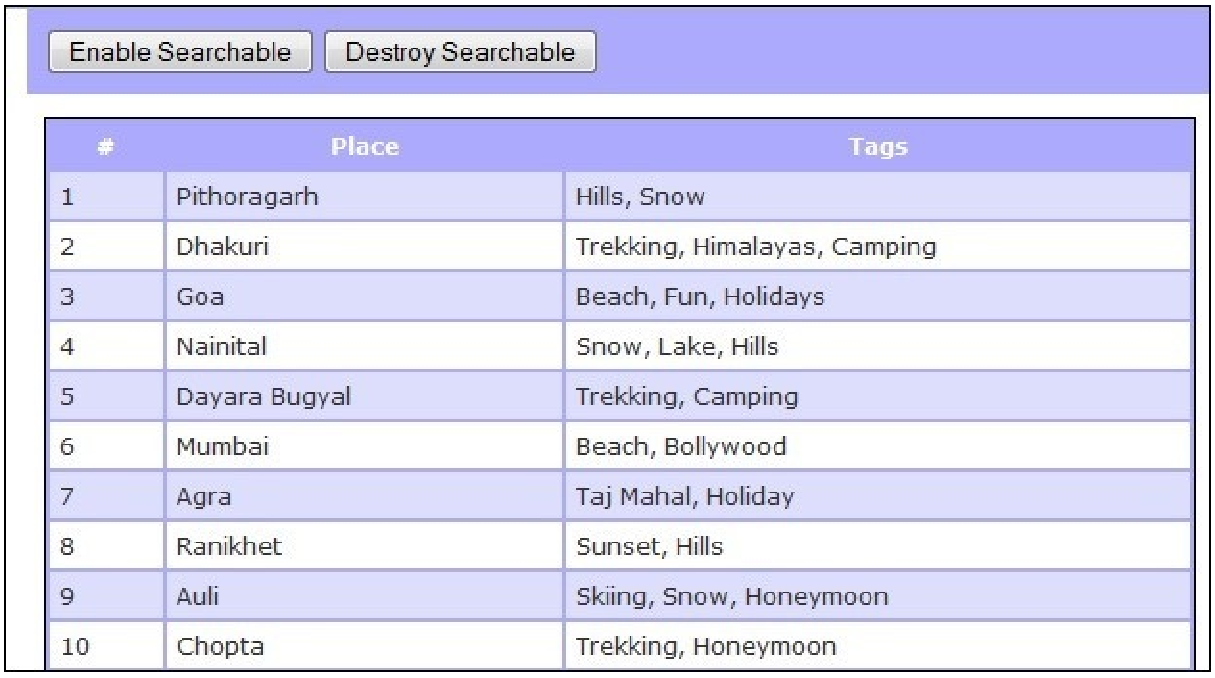
We will do some basic styling to make the table look better. We will apply styles for the `table` and `td` elements. There will be an interesting addition, though. We will also define some CSS rules for classes of elements that will be generated by the widget. Here are the CSS rules to apply in the `head` section of the file:

```
<style type="text/css">
  body{
    margin:0 auto;
    font-family:verdana,arial;
    font-size:12px;width:45%
  }
  .tables{
    border:1px solid #000;
    margin:0 auto;
    width:600px;
    background-color:#acaafc;
  }
  .tables th{
    color:#fff;
  }
  th,td{
    padding:5px;
    font-size:12px;
  }
  p{
    background-color:#acaafc;
    padding:10px;
  }
  .even{
    color:#343234;
    background-color:#fff;
  }
  .odd{
    color:#343234;
    background-color:#dcdefc;
  }

/* styles specific to search widget */
  .mywidget-searchBoxContainer{
  }
  .mywidget-searchBoxContainer td{
    border:1px solid #fff;
  }
  .mywidget-label-search{
    color:#fff;
  }
  .mywidget-textbox{
  }
</style>
```

Note the four CSS rules at the end of the code. When the widget generates a new row, the `mywidget-searchBoxContainer` class will be applied to the row. For the widget's `td` element, the `mywidget-searchBoxContainer td` rule will be applied. The label and the textbox will have the `mywidget-label-search` and `mywidget-textbox` classes applied to them, respectively.

The purpose of applying these CSS rules in widget's HTML code is to allow theming. You can place any CSS properties for these to customize the look of the HTML generated by the widget.

If you load the `search.html` file in the browser now, you will see a nice-looking table and two buttons at the top, as shown in the following screenshot:



#	Place	Tags
1	Pithoragarh	Hills, Snow
2	Dhakuri	Trekking, Himalayas, Camping
3	Goa	Beach, Fun, Holidays
4	Nainital	Snow, Lake, Hills
5	Dayara Bugyal	Trekking, Camping
6	Mumbai	Beach, Bollywood
7	Agra	Taj Mahal, Holiday
8	Ranikhet	Sunset, Hills
9	Auli	Skiing, Snow, Honeymoon
10	Chopta	Trekking, Honeymoon

Implementing the widget

The widget factory follows a common pattern to define its methods. There are some basic methods that must be implemented in order to create a widget. Hence, we will first define the structure of the widget, and then understand the widget in detail to see what each of the methods do, and how we can use these methods to create widgets of our own.

Let’s look at the widget structure first.

Defining the widget structure

The following code represents the life cycle of a typical jQuery UI widget. There are many more properties that are not used so often. We will look at those properties at the end of this chapter.

For now, write the following code in the `js/searchable.js` file to create the basic skeleton of the widget:

```
(function ($) {
  $.widget("mywidget.searchable" ,
  {
    options:
    {
        characterLength: 3,
          searchLabel : 'Enter characters : '
    },
    _create: function ()
    {
    },
    _destroy: function ()
    {
    },
    _setOption: function (key, value)
    {
    }
  });

})(jQuery);
```

The widget is defined using `$.widget()` that takes two arguments: the first argument is the name of the widget with the namespace and the second argument is the set of properties of the widget. We set the name as `mywidget.searchable`. Here, `mywidget` is the namespace and `searchable` is the widget name by which it will be initialized by other users. All the widgets of jQuery UI share the namespace `ui`. Hence, the `dialog` widget is created using `ui.dialog` and so on. You can assign any namespace to your widgets.

After the widget name, we defined the properties for the widget. The preceding code contains the basic properties that are required by `$.widget`. Let’s look at all of these:

- `options`: This is an object that has the key-value pairs. The keys and values provided here serve as the default `options` values for the widget. These keys will be provided by users when the widget is initialized.
- `_create`: This is the constructor of the widget. We will create the DOM required for

the widget inside this method and inject it into the page. Event handlers for the widget's elements will also be added here.

- `_destroy`: This method is invoked when the `destroy` option is called on the method. Calling the `destroy` method invokes a public `destroy` method of the jQuery UI, which removes any event handlers and data present in DOM. It then passes control to `_destroy`, where we can remove the widget's HTML from the page and revert any changes that the widget made to the DOM.
- `_setOption`: Whenever any option is set for the widget by using the `option` method, a method named `_setOptions` is called (note the s at the end). This method calls the `_seOption` method for each option provided by the user. This method receives the `option key` and `value`. Depending on the value of a particular key, changes are made to the widget state, as required.

Now, we are ready to implement the code for various widget methods.

Setting up default options

For this example, we have defined two options, `characterLength` and `searchLabel`, and set the values to `3` and `Enter characters :`, respectively.

The value of the `characterLength` option is the minimum number of characters the user has to type before we search the table for input value. The value `3` means the widget will search the table only when user has typed at least three characters.

The `searchLabel` option will have the text that will appear towards the left of the textbox. Users will be able to set both these options when they initialize the widget.

Initializing the widget and attaching event handlers

After setting up the options, we will now implement the widget constructor `_create`, where we will create the DOM and attach the event handler for searching.

Go to the `_create` method in the `searchable.js` file and write this code:

```
if(!this.element.is('table'))
  {
    console.log('not a table');
    return;
  }

  this.element.addClass('mywidget-searchable-table');

  var colspan = (this.element).find('tr:first').children().length;

  this.searchInput = $("<input type='text' class='mywidget-textbox ui-
state-highlight ui-corner-all'>")
.insertBefore((this.element).find('tr:first'))
  .wrap('<tr class="mywidget-searchBoxContainer"><td colspan="'+colspan+'">
</td></tr>');

  $("<label class='mywidget-label-search'>"+ this.options.searchLabel+"
</label>").insertBefore(this.searchInput);
```

```
  this._on(this.searchInput,
  {
    keyup: "_filterTable"
  }
);
```

Note

The `this.element` and `this.options` will be available inside all methods. The `this.element` refers to the element on which the widget was called and the `this.options` allows you to access any option of the widget.

Since we want this widget exclusively for tables, the first thing we check inside the `_create` method is that `this.element` must be a table element. If it is not, we log an error in the console and exit from the method.

Next, we added a CSS class `mywidget-searchable-table` to the table. You can write any CSS for this class to customize the look of the widget. Then, we find the number of cells in a particular row. This will be used to set the `colspan` value of the row we are going to append to the table.

Following the code is very important, as we will create the HTML and append it to DOM. First, we created an input box with the class `mywidget-textbox`. Then, we inserted the input box before the first `tr` element of the table. Finally, we created a new `tr` element with a `td` element inside it and wrapped the newly created element inside this new row. The `colspan` value of the `td` is set to the variable `colspan` we calculated earlier. Also, note that the `tr` has been assigned a CSS class `mywidget-searchBoxContainer`. The newly created element will be available via `this.searchInput` object to rest of the widget methods.

We also need to place a label towards the left of the input box. Therefore, we create a label element, assign the CSS class `mywidget-label-search` to it and insert it before the newly created search box that is accessible using `this.searchInput`. Inside the `label`, we set its text to the value of `searchLabel` option.

In order for the textbox to be functional, we need to add an event handler. We will attach a `keyup` event handler. This is done using the `_on` method of the widget factory. On the, `keyup` event, a method named `_filterTable` will be called where we will filter the table rows.

Filtering the table

Inside the `searchable.js` file, we will have to add a new method named `_filterTable` first and then implement the required code. The following code defines the `_filterTable` method and the code to search all the cells of table:

```
_filterTable: function (event)
{
  var inputVal = $.trim(this.searchInput.val());
  if(inputVal.length < this.options.characterLength)
  {
    this.element.find('tr').show();
```

```
      return;
    }
    this.element.find('tr:gt(0)').each(function(index,row)
    {
      var found=false;
      $(row).find('td,th').each(function(index,td)
      {
        var regExp=new RegExp(inputVal,'i');
        if(regExp.test($(td).text()))
        {
          found = true;
        }
      });
      if(found)
      {
        $(row).show();
      }
      else
      {
        $(row).hide();
      }
    });
  }
```

Note

If you are adding this method after any of the other methods in the `searchable.js` file, make sure that you add a comma (,) to separate different methods.

We begin by storing the value of the input box in a local variable `inputVal`. Then, we check the length of input against the `characterLength` option. If the provided text input is less than the `characterLength` option, we simply display all the `tr` elements and exit. We proceed only if the input length is more than the value of the `characterLength` option.

Next is a loop using `jQuery's $.each` to iterate in all rows. Note that the `tr:gt(0)` selector used for the `tr` elements. Since the first `tr` element is the search widget itself, we collected rows starting from the index '1'. For each `tr` element in the loop, we receive the index of that row and the row element itself.

For each row, we set a flag named `found` to `false`. Then, we loop again in the `td` and `th` elements of that particular row. For each row, we test the input value against a regular expression. If the input value matches the text in any `td` element of that row, we set the `found` flag to `true`.

Once looping over `td` and `th` elements is complete, we show or hide the rows based on the value of `flag` found.

With this implementation, we are halfway through creating the widget. Let's implement the remaining methods now.

Making changes when the option value is changed

Once the widget has been initialized, users should be able to change the `characterLength` and `searchLabel` options using the option method. In case of the `searchLabel` option, we will have to display the new text in the page as well. The following code will take care of both options:

```
_setOption: function (key, value)
{
  switch (key)
  {
    case "searchLabel":
      this.searchInput.prev('label').text(value);
      break;
    default:
      break;
  }
  this.options[ key ] = value;
  this._super("_setOption", key, value);
}
```

Since `_setOption` is called for each option, we need to identify the keys and take actions accordingly. For this purpose, we have placed a `switch` block with each option name as a `case` condition.

The first case is for the option `searchLabel`. If this option is set, we change the text of label to the new value. Also, we set the new value for each option using the following code:

```
this.options[ key ] = value;.
```

Lastly, the `_super` method is called for `_setOption`. This method is a jQuery UI method that updates the state of the widget internally.

Destroying the widget

The last task is to destroy the widget and clean the DOM elements we have created. The following code will remove the widget HTML and revert the table to its original state:

```
_destroy: function ()
{
  this.element.removeClass('mywidget-searchable-table');
  $('.mywidget-searchBoxContainer').remove();
}
```

The first line removes the theming class `mywidget-searchable-table` from the table, and the second line removes the `tr` element from the table.

Remember that after calling the `destroy` method for the widget, jQuery UI will call its internal `destroy` method, which will unbind all event handlers as well.

Calling the widget from the page

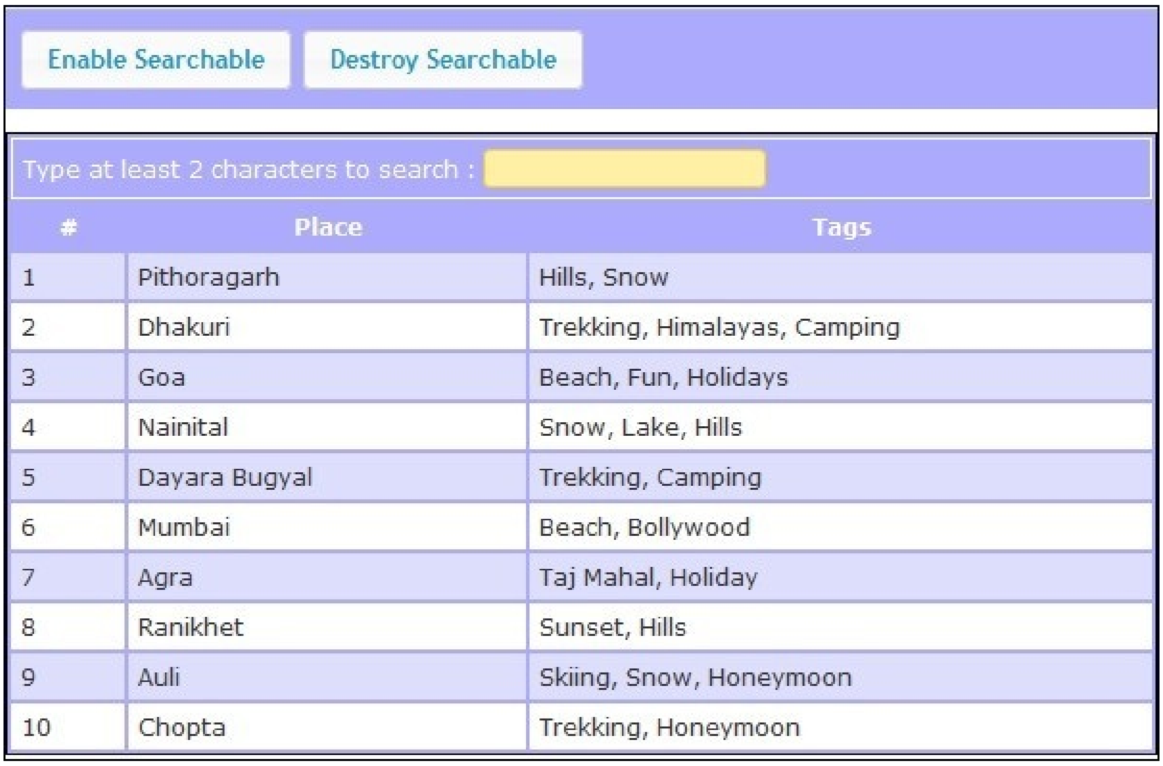
Now that we have implemented all methods, our `searchable` widget is complete and we are now ready to use it:

1. Open the `search.html` file.
2. Go to the bottom where you have included the `searchable.js` file. After `searchable.js` is included, insert the following code to bind event handlers for the **Enable Searchable** and **Destroy Searchable** buttons:

```
<script type="text/javascript">
  $(document).ready(function()
  {
    $('#btnEnable').on('click', function()
    {
      $('#tblData').searchable(
      {
        characterLength : 2,
        searchLabel : 'Type at least 2 characters to search : '
      });
    });

    $('#btnDestroy').on('click', function()
    {
      $('#tblData').searchable('destroy');
    });

  $('button').button();
  });
</script>
```

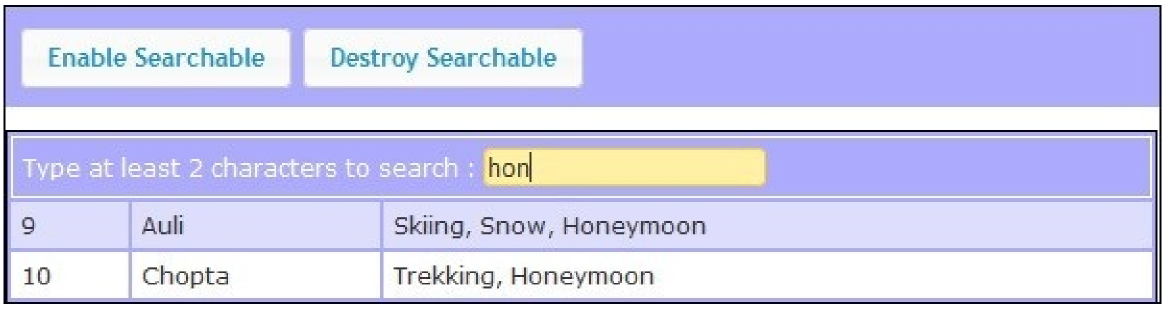
   The code here is pretty self-explanatory. We have bound the click event handlers for buttons with `id` values `btnEnable` and `btnDestroy`. On a click of `btnEnable`, we initialize the `searchable` widget with the options `characterLength` and `searchLabel`. We have set the `characterLength` option to `2` here. For `btnDisable`, we call the `destroy` method that will remove the widget completely.

3. Then, we will call the `button` widget on the buttons present in the page.
4. Open the `search.html` file in a browser now and click on the `Enable Searchable` button. The widget will appear on the page:



#	Place	Tags
1	Pithoragarh	Hills, Snow
2	Dhakuri	Trekking, Himalayas, Camping
3	Goa	Beach, Fun, Holidays
4	Nainital	Snow, Lake, Hills
5	Dayara Bugyal	Trekking, Camping
6	Mumbai	Beach, Bollywood
7	Agra	Taj Mahal, Holiday
8	Ranikhet	Sunset, Hills
9	Auli	Skiing, Snow, Honeymoon
10	Chopta	Trekking, Honeymoon

5. Now type a few characters in the search box. We have initialized the widget with `characterLength` option set to `2`. As soon as you type the second character, filtering of rows will start with each keystroke. Here is how the page will look after filtering:



9	Auli	Skiing, Snow, Honeymoon
10	Chopta	Trekking, Honeymoon

Improving the searchable widget

The `searchable` widget we made is a basic one but you must have got the idea of providing options and adding events. You can try the following suggestions to enhance this widget:

- Option to append the widget at either top or bottom
- Case-sensitive search
- Excluding specific text from the search

Creating a widget to display a slideshow

In the previous widget, we did not make use of callbacks. Most of the widgets require some type of custom callbacks. They provide more control to users and allow users to interact with the widget; for example, the built-in dialog. It provides options such as open, close, create, drag, and so on, which can be used to add dynamic behavior to the widget. Therefore, for our second widget, we will create the widget options as well as a callback. This will help you to understand widgets more thoroughly.

We will address this problem in our second widget, where we will convert a list of elements into slideshow. We will allow users to customize the widget by providing options to set the duration of a slide, the effect which will be used to show and hide the slides, the speed of slides, and so on. We will also provide a callback method. This callback will be called just before a slide is displayed.

Writing markup for slides

Let's start by writing the markup for the page. For the markup, we will create a `div` element that will act as a parent container, and the child elements of this `div` will act as individual slides. For our example, we will keep an image and some text in each slide.

To write the markup, open the `slides.html` file and write the following code to create the structure:

```
<html>
  <head>
    <title>Slideshow</title>
    <link rel="stylesheet" href="css/ui-lightness/jquery-ui-
1.10.4.custom.min.css">
  </head>
  <body>
  <p>
    <button id="btnEnable">Start Slideshow</button>
    <button id="btnDestroy">Stop Slideshow</button>
  </p>
    <div id="slideContainer">
      <div class="slide ui-state-highlight">
        <img src="images/1.jpg">
        <p>Cras congue nisl in tellus placerat luctus. Mauris tempus ante
erat, non tempus enim posuere vel. In condimentum orci sem, a vestibulum
leo elementum eu. </p>

      </div>
      <div class="slide ui-state-highlight special">
        <img src="images/2.jpg">
        <p>Ut interdum, massa quis feugiat consectetur, enim ligula varius
mi, vitae varius massa elit quis velit. In posuere egestas velit ac
molestie. Vestibulum nec dapibus justo. </p>

      </div>
      <div class="slide ui-state-highlight">
        <img src="images/3.jpg">
        <p>Morbi posuere molestie mauris a ornare. Integer at ipsum vel
metus rutrum suscipit at nec ante. Nullam malesuada tempor elementum. Nam
nec sollicitudin massa. Pellentesque maximus diam at libero faucibus
porttitor. </p>

      </div>
      <div class="slide ui-state-highlight">
        <img src="images/4.jpg">
        <p>In commodo laoreet mi, congue placerat purus. Cras a feugiat
velit. Nunc facilisis ac tortor a consequat. Integer congue purus et
hendrerit volutpat. Duis dictum malesuada placerat. </p>

      </div>
    </div>
    <script src="js/jquery-1.10.2.js"></script>
    <script src="js/jquery-ui-1.10.4.custom.min.js"></script>
    <script src="js/slides.js"></script>
  </body>
```

```
</html>
```

We started off by including the jQuery UI CSS file in the page. Inside the body section, we created two buttons with `id` values `btnEnable` and `btnDestroy`, just like the previous widget. These buttons will enable and disable the slideshow on clicking.

After buttons, there is a `div` with the `id` value `slideContainer`. Inside this `div`, we have created four more `div` elements. Each of these will be a slide. Each of these div elements has been given the classes `slide` and `ui-state-highlight`. Every slide also has an image and some text inside it. All the images have been referenced from the images directory inside the `Chapter9` directory.

Note that the second `div` has another CSS class named `special` assigned to it. We will use it later with the callback method.

Finally, just before the closing of the `body` tag, we include jQuery, jQuery UI files, and the `slides.js` file that is empty at the moment.

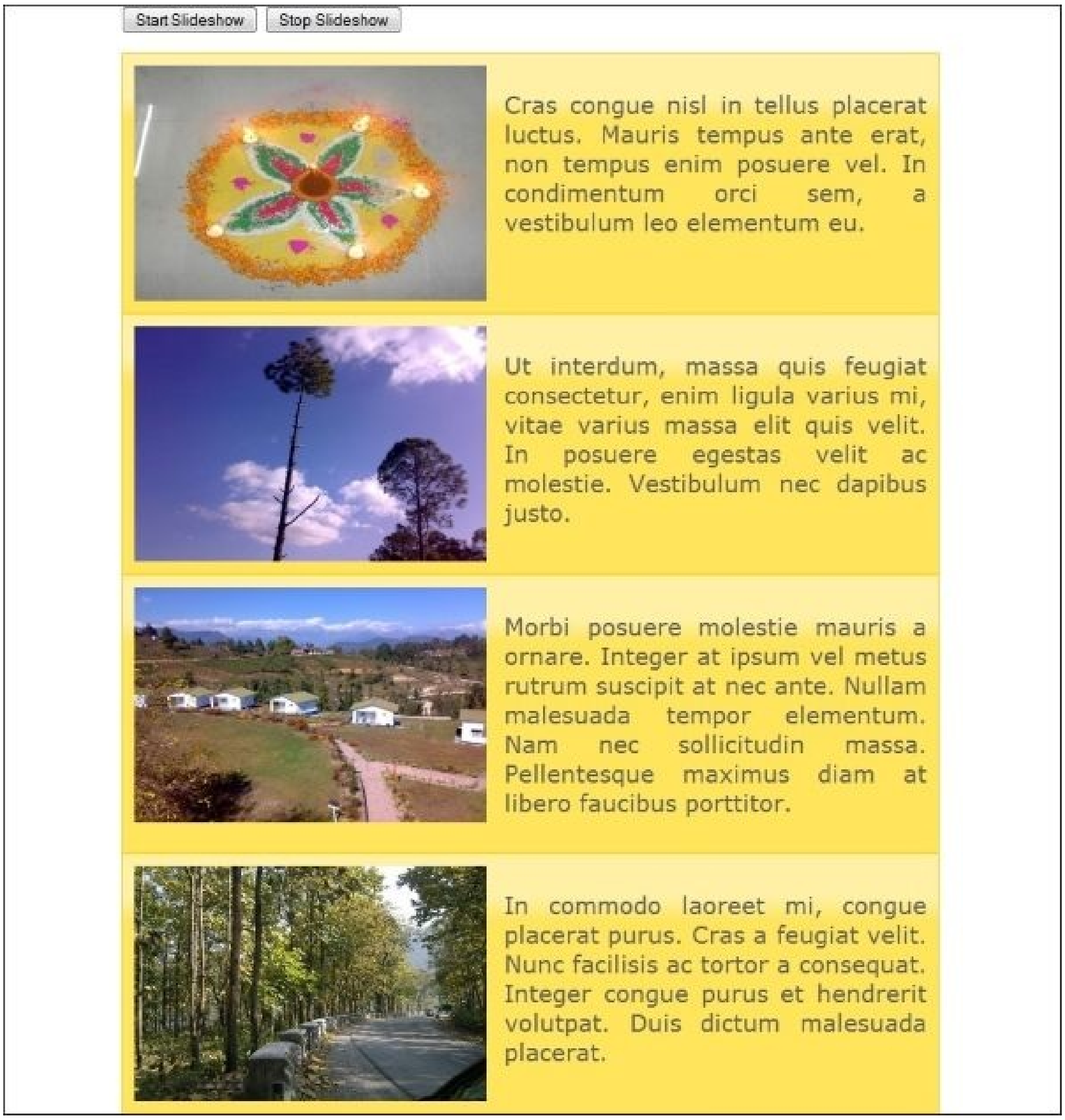
Styling the content

Before we check the page in browser, let's beautify it a bit. We will define some CSS properties for `div#slideContainer`, `div.slide`, and the `p` and `img` elements inside it. Go to the `head` section of `slides.html` and write the following CSS rules after the jQuery UI CSS file is included:

```
<style type="text/css">
  body{
    color:#025c7f;
    font-family:verdana,arial;
    width:700px;
    margin:0 auto;
  }
  #slideContainer{
    margin:0 auto;
    font-size:20px;
    position:relative;
    width:700px;
    text-align:justify;

  }
  .slide{
    float:left;
    padding:10px;
  }
  .slide img{
    height: 200px; width: 300px; float: left;
  }
  .slide p{
    display: inline-block; width: 360px; margin-left: 15px;color:#5f5f5f;
  }
  a{
    color:#000;
    font-size:15px;
  }

  /* styles specific to banner rotator */
  .mywidget-banner{
  }
  .mywidget-banner-item{
  }
</style>
```

The last two CSS rules are for the class names that will be generated by the widget. You can specify the CSS properties here and they will be applied when the widget is activated.

Now we can see what the page looks like. Open up your browser and load the `slides.html` file. You will see the slides in a column:



Cras congue nisl in tellus placerat luctus. Mauris tempus ante erat, non tempus enim posuere vel. In condimentum orci sem, a vestibulum leo elementum eu.

Ut interdum, massa quis feugiat consectetur, enim ligula varius mi, vitae varius massa elit quis velit. In posuere egestas velit ac molestie. Vestibulum nec dapibus justo.

Morbi posuere molestie mauris a ornare. Integer at ipsum vel metus rutrum suscipit at nec ante. Nullam malesuada tempor elementum. Nam nec sollicitudin massa. Pellentesque maximus diam at libero faucibus porttitor.

In commodo laoreet mi, congue placerat purus. Cras a feugiat velit. Nunc facilisis ac tortor a consequat. Integer congue purus et hendrerit volutpat. Duis dictum malesuada placerat.

Looks like our HTML is ready and we are prepared to spice it up with some jQuery goodness. Let’s start by implementing the widget now.

Implementing the widget

Like the previous widget, we will begin by defining the structure of the widget first. Since we have already covered the various methods in previous widgets in detail, we will not go through them here.

Defining the widget structure

Open the `js/slides.js` file and create the structure of the widget, as shown in the following code:

```
(function ($) {
  $.widget("mywidget.slideshow" ,
  {
    options:
    {
      duration: 3000,
      effect : 'bounce',
      easingDuration : 'slow',
      beforeShow : null
    },
    _create: function ()
    {
    },
    _destroy: function ()
    {
    },
    _setOption: function (key, value)
    {
    }
  });

})(jQuery);
```

We defined the four basic properties required to create a widget: the `options` object, the constructors `_create` and `_destroy`, and the option-setter method `_setOption`.

Setting up default options

We have provided four options that could be customized by users:

- `duration`: This is the time in milliseconds for which a slide will be displayed. The default value is `3000`.
- `effect`: This is the effect that will be used to show and hide the slides. The default value is `bounce`. Any of the jQuery easing names can be used here.
- `easingDuration`: This is the duration of effect in milliseconds.
- `beforeShow`: This is the callback method that users will be able to override. Currently, it has been set to `null`.

Users will be able to override any of these options while initializing the widget. If any `option` value is not provided during initialization, jQuery UI will use the default value as defined in widget options.

Initializing the widget and displaying the first slide

We can now start defining the constructor method `_create`, where we will initialize the widget. This means we can add CSS classes to elements and display the first slide.

Write the following code for the `_create` method in the `js/slides.js` file:

```
_create: function ()
{
  this.element.addClass('mywidget-banner');
  this.element.children().addClass('mywidget-banner-item');
  $('.mywidget-banner-item').hide();

  this._trigger( "beforeShow", null , { element : $('.mywidget-banner-
item:first')} );

  $('.mywidget-banner-item:first')
  .addClass('current')
  .show(this.options.effect, this.options.easingDuration);
  this._setRotation();
}
```

As you now know, `this.element` refers to the element that was passed while initializing the widget, which will be `div# slideContainer` in this case. In the first line, we attach a CSS class `mywidget-banner` to to that element.

The next line attaches the CSS class `mywidget-banner-item` to each of the slide divs. Then, we hid all slides using the jQuery `hide` method.

Now we can display the first slide with animation. However, first we will have to invoke the `beforeShow` callback that has to be fired just before a slide is displayed. This is done with the help of `_trigger`, a method provided by the widget factory.

As the name suggests, `_trigger` triggers the callback that is passed to it. It takes three parameters: the event name, then the event that triggered this event, and the data that we want to pass to the event. In our case, we have passed the element that will be available in the form of the `element` property inside the callback.

After the callback event has been triggered, we can now display the first slide. We do this by selecting the first slide using the `.mywidget-banner-item:first` selector. Then, we add a `current` class to it. This is necessary to find out which element is currently being displayed.

Note

Instead of the `current` class, we can also use a variable internally and update it each time a slide is displayed.

After adding the class, we use the `show` method to display the first slide. Note that we have passed the widget options `effect` and `easingDuration` to show the method.

Our task is only half done yet. We also need a way to go through all the slides one by one. For this purpose, we need to set up a timer which will execute repeatedly at the specified duration. The code in the last line is used for the same purpose. We have also called a `_setRotation` method, which we will explore in the next section.

Displaying slides one by one

After the first slide is displayed, a call was made to `_setRotation` in order to set the time. We will use the JavaScript method `setInterval`, which will execute a callback function at fixed intervals. Here is the definition of the `_setRotation` method that we will create inside the widget:

```
_setRotation : function()
{
  var that = this;
  this.interval = setInterval(function()
  {
    that.textRotate();
  },that.options.duration);
}
```

Inside `_setRotation`, we stored the reference to local scope in a local variable named `that`. Then, we called the JavaScript method `setInterval`. As you can see in the preceding code, `setInterval` will call another method `textRotate` (which we will define next). We also defined the duration after which `textRotate` will be called by reading the duration value from the `options` object.

Note

Note the variable `this.interval`. This will be used again when user changes the `duration` option.

Now, we need to define the `textRotate` method that will take care of going through the slides. Once we reach the last slide, we will restart from first slide. In the following code, we will define the `textRotate` method after the `_setRotation` method:

```
textRotate : function()
{
  var $that = this;
  var current = $that.element.find('.current');
  var next = current.next();
  if(next.length==0)
  {
    current.removeClass('current').hide($that.options.effect,
$that.options.easingDuration, function()
    {
      $('.mywidget-banner-
item:first').addClass('current').show($that.options.effect,
$that.options.easingDuration);
    });
  }
  else
  {
    current.removeClass('current').hide($that.options.effect,
$that.options.easingDuration, function()
    {
      $that._trigger( "beforeShow", null, { element : next} );
      next.addClass('current').show($that.options.effect,
$that.options.easingDuration);
    });
  }
}
```

Have patience if the preceding code looks a bit cryptic. Let's understand it line by line:

- First, we create a local variable `$that` and store the reference to current scope variable `this` in it. After this, we store the reference to the element currently being displayed in the variable `current`.

  ### Note

  The CSS class `current` is applied on the slide which is currently visible.

- Then, we find the elements next to the element with CSS class `current`. If the `current` class is on the last slide, the value of `next` will be 0; otherwise, it will be greater than 0.
- The `if` condition in the following line checks the value of the variable `next`. If the value is 0, it means that currently the last slide is being displayed and we need to show the first slide after this. In this case, we remove the CSS class `current` from the current slide and hide it. Three parameters are passed to the `hide` method: first is the effect used to hide the slide, second is the duration of hiding effect animation, and the third parameter is the callback function that will be called once `hide` method completes the animation.
- The third parameter is used to display the next slide (in this case, the first slide). To select the first slide, we simply use the selector `.mywidget-banner-item:first`, add the class `current` to it, and use the `show` method with the `effect` and `easingDuration` values.
- For slides other than the last one, the `else` block will be executed. Here as well, we remove the current class from the currently visible slide and hide it with the `effect` and `easingDuration` options. In the callback of the `hide` method, we trigger the `beforeShow` callback so that the user may introduce any custom behavior. Then, we add the `current` class to the `next` element which is stored in the `next` variable, and call the `show` method. Now, we will pass the effect and the duration of effect from the `options` object.

That is all we need to make the slides functional. You might be tempted to check the progress so far, but we have a couple of things to do before that. Handling the change in any of the `option` values, and cleaning up after the `destroy` method is called.

Making changes when the option value is changed

The `option` method can be used on any widget to get or set the value of widgets. If a user sets a new value for an option, there should be a way for the widget to know it and change the widget state accordingly.

We will use the `_setOption` method in the same way we did in the previous widget. A `switch` case will be applied for all the options. Write the following code for `_setOption`, which will help maintain the state of the widget:

```
_setOption: function (key, value)
{
  switch (key)
  {
    case "duration":
      clearInterval(this.interval);
      this.options[ key ] = value;
      this._setRotation();
      break;
    default:
      this.options[ key ] = value;
      break;
  }
  this._super("_setOption", key, value);
}
```

For options such as `effect` and `easingDuration`, there is no behavioral change in the widget. We just need to update the values and the widget will pick up the latest values. However, it is different for the `duration` option. In this case, we cleared the previous interval using the JavaScript method `clearInterval`. Then, we will reset the value for the `duration` option and finally call the `_setRotation` method again, so that the `setInterval` method will be called using the new `duration` value.

So, now at any point the user can change the duration of slide using duration option.

Destroying the widget

To destroy the widget, we will have to stop the repeated calls to the `textRotate` method and remove all the CSS classes that we attached earlier in the `_create` method. Here is the code:

```
_destroy: function ()
{
  clearInterval(this.interval);
  this.element.removeClass('mywidget-banner');
  this.element.children().removeClass('current mywidget-banner-
item').show();
}
```

We start by removing the repeated call to `textRotate` by removing the interval using the `clearInterval` method.

Then, we remove the `mywidget-banner` class that was applied to the `slider` container. Then, we remove the `mywidget-banner-item` class from all the items and the `current` class as well. The `show` method in the end ensures that all the slides are visible, as they were prior to calling the widget.

This finishes our slideshow widget and we are now ready to see it in action.

Calling the widget from the page

To call the widget, we will have to initialize it in the `slides.html` file. Hence, go to this file and write the following code to enable and disable slideshow on clicking **Start Slideshow** and **Stop Slideshow** buttons, respectively:

```
<script type="text/javascript">
$(document).ready(function()
{
  $('#btnEnable').on('click', function()
  {

    $('#slideContainer').slideshow(
    {
      duration : 4000,
      effect : 'clip',
      easingDuration : 400,
      beforeShow : function(event, ui)
      {
        if(ui.element.hasClass('special'))
        {
          ui.element.css({ 'background' : '#ffffff' });
        }
      }
    });
  });

  $('#btnDestroy').on('click', function()
  {
    $(slideContainer).slideshow('destroy');
  });

  $('button').button();
});
</script>
```

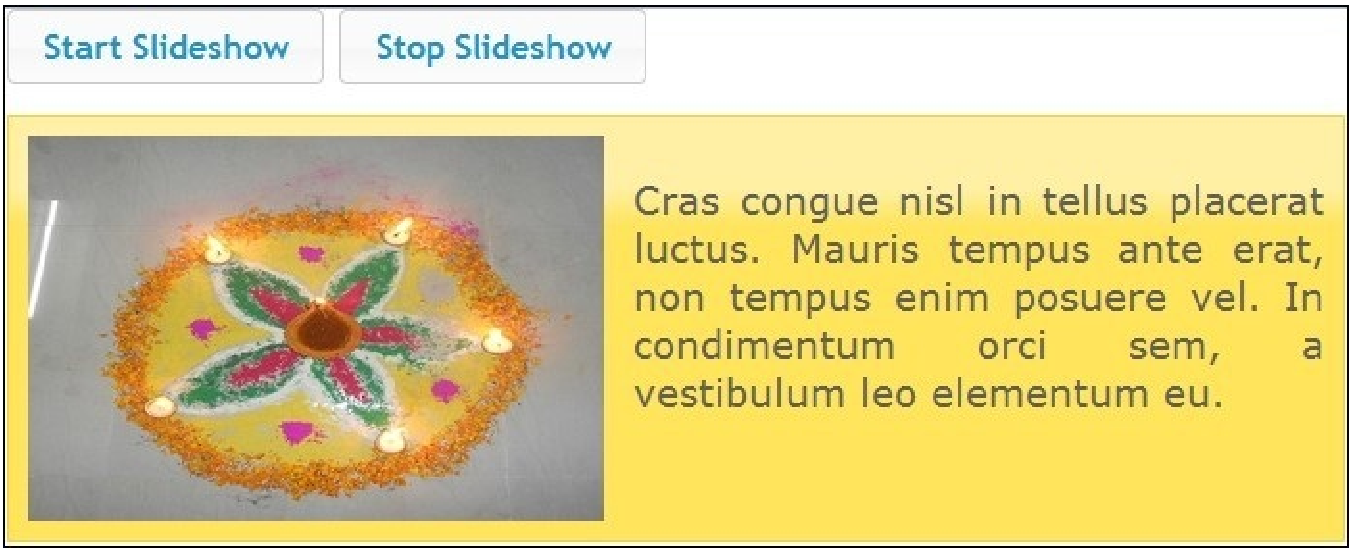
Firstly, we attach an event handler on click of first button with `id` value `btnEnable`. Inside the event handler, we initialize the slideshow widget with three options and a callback. We have provided the `duration` as `4000` milliseconds, `effect` as clip, and `easingDuration` as `400` milliseconds.

We can also use the callback `beforeShow` now. If you recall, we placed a CSS class `special` in the second slide. Since this callback is triggered every time just before a slide is displayed, we are checking if the slide `div` has a class `special` attached to it. For such an element, we set the `background` to white.

Next, we attach the event handler for the `destroy` event. In the event handler, we just call the `slideshow` widget with destroy options.

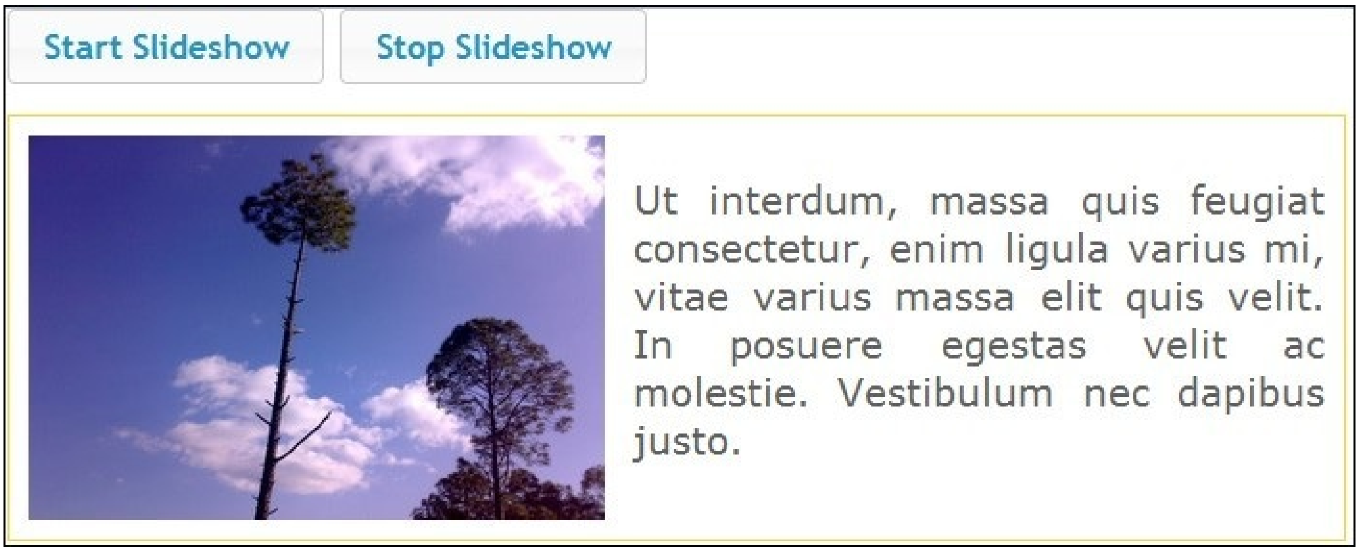
Finally, we convert the enable and disable slideshow buttons to jQuery UI buttons.

We can now check the widget in action. Browse the `slides.html` file in the browser and you will see all slides in a column. Clicking the **Start Slideshow** button will begin the slideshow and the first slide will be displayed with the provided effect:



Cras congue nisl in tellus placerat luctus. Mauris tempus ante erat, non tempus enim posuere vel. In condimentum orci sem, a vestibulum leo elementum eu.

On the second slide, you will also see the effect as the background will change to white due to the `beforeShow` callback:



Ut interdum, massa quis feugiat consectetur, enim ligula varius mi, vitae varius massa elit quis velit. In posuere egestas velit ac molestie. Vestibulum nec dapibus justo.

Clicking on the **Stop Slideshow** button will reset the page to its initial state and remove the widget behavior completely.

Improving the banner widget

Here are a few suggestions to help you add new features to the slideshow widget:

- Add different effects to show and hide a slide
- Implement another callback `afterHide` that will be called after a slide is hidden
- Disable the slideshow after a certain number of iterations of all slides

Summary

We created two different widgets in this chapter, where you learned to customize the widget with the help of options. You also learned to implement custom callbacks for the widget as well. All in all, we covered the full life cycle of a widget. With the help of this chapter, you will be able to create your own widgets with ease.

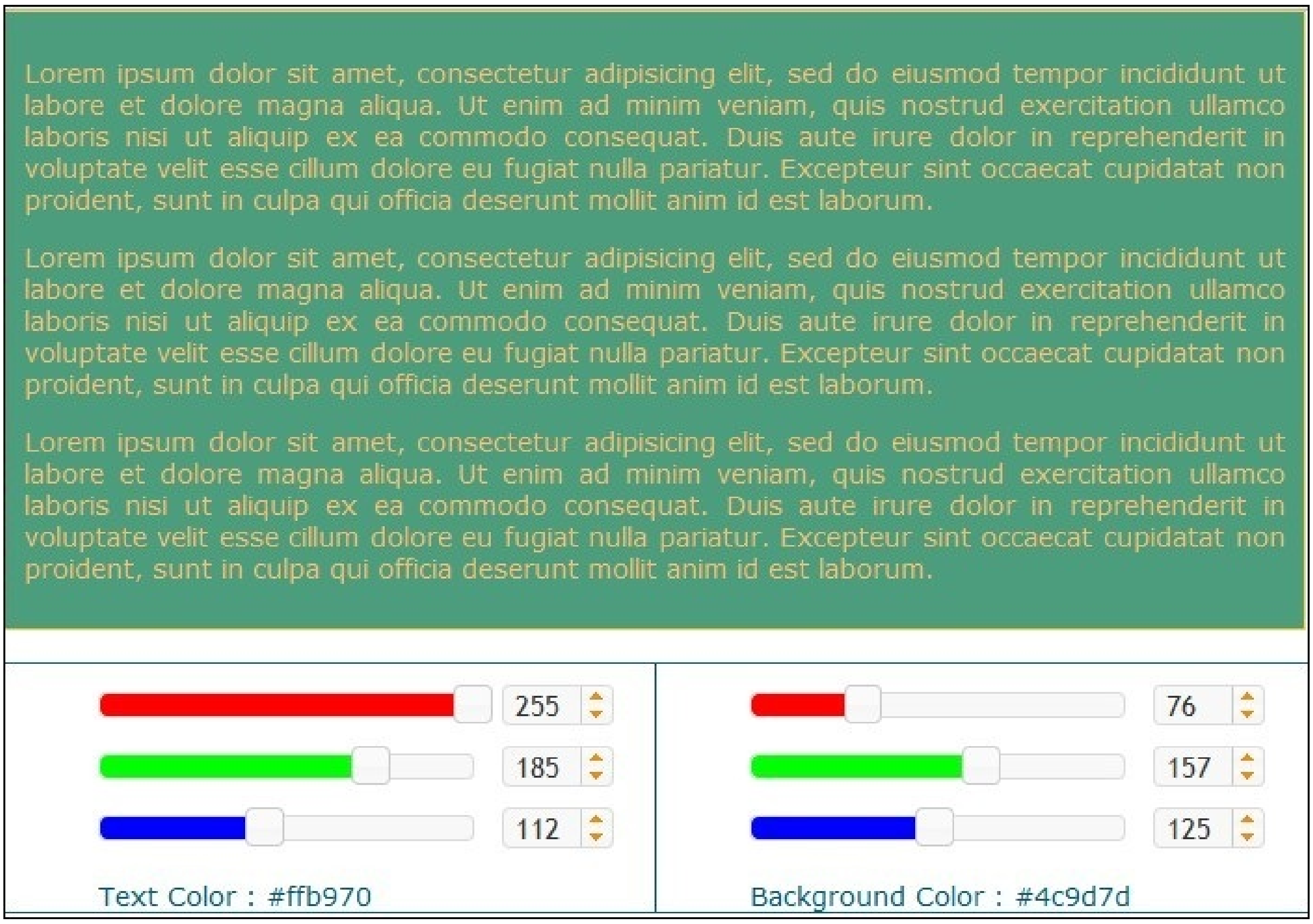
Moving forward, in the next chapter, you will learn to create a “colorpicker” tool and you will also allow users to convert RGB values of colors to Hex and vice versa.

Chapter 10. Building a Color Picker with Hex RGB Conversion

Chapter 9, *Creating Widgets Using the Widget Factory*, was a complex one where we created two widgets using jQuery UI's widget factory. In this chapter, we will relax a bit and build something simple.

We are going to create a color selector, or color picker, that will allow the users to change the text and background color of a page using the slider widget. We will also use the spinner widget to represent individual colors. Any change in colors using the slider will update the spinner and vice versa. The hex value of both text and background colors will also be displayed dynamically on the page.

This is how our page will look after we have finished building it:



Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

255
185
112
Text Color : #ffb970

76
157
125
Background Color : #4c9d7d

Setting up the folder structure

To set up the folder structure, follow this simple procedure:

1. Create a folder named `Chapter10` inside the `MasteringjQueryUI` folder.
2. Directly inside this folder, create an HTML file and name it `index.html`.
3. Copy the `js` and `css` folder inside the `Chapter10` folder as well.
4. Now go inside the `js` folder and create a JavaScript file named `colorpicker.js`.

With the folder setup complete, let's start to build the project.

Writing markup for the page

The `index.html` page will consist of two sections. The first section will be a text block with some text written inside it, and the second section will have our color picker controls. We will create separate controls for text color and background color. Inside the `index.html` file write the following HTML code to build the page skeleton:

```
<html>
  <head>
    <link rel="stylesheet" href="css/ui-lightness/jquery-ui-
1.10.4.custom.min.css">
  </head>
  <body>
    <div class="container">
      <div class="ui-state-highlight" id="textBlock">
        <p>
          Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do
eiusmod
          tempor incididunt ut labore et dolore magna aliqua. Ut enim ad
minim veniam,
          quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea
commodo
          consequat. Duis aute irure dolor in reprehenderit in voluptate
velit esse
          cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat
cupidatat non
          proident, sunt in culpa qui officia deserunt mollit anim id est
laborum.
        </p>
        <p>
          Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do
eiusmod
          tempor incididunt ut labore et dolore magna aliqua. Ut enim ad
minim veniam,
          quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea
commodo
          consequat. Duis aute irure dolor in reprehenderit in voluptate
velit esse
          cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat
cupidatat non
          proident, sunt in culpa qui officia deserunt mollit anim id est
laborum.
        </p>
        <p>
          Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do
eiusmod
          tempor incididunt ut labore et dolore magna aliqua. Ut enim ad
minim veniam,
          quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea
commodo
          consequat. Duis aute irure dolor in reprehenderit in voluptate
velit esse
          cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat
cupidatat non
```

```
          proident, sunt in culpa qui officia deserunt mollit anim id est
laborum.
        </p>
      </div>

      <div class="clear">&nbsp;</div>

      <ul class="controlsContainer">
        <li class="left">
          <div id="txtRed" class="red slider" data-spinner="sptxtRed" data-
type="text"></div><input type="text" value="0" id="sptxtRed" data-
slider="txtRed" readonly="readonly" />
          <div id="txtGreen" class="green slider" data-spinner="sptxtGreen"
data-type="text"></div><input type="text" value="0" id="sptxtGreen" data-
slider="txtGreen" readonly="readonly"  />
          <div id="txtBlue" class="blue slider" data-spinner="sptxtBlue"
data-type="text"></div><input type="text" value="0" id="sptxtBlue" data-
slider="txtBlue" readonly="readonly"  />

          <div class="clear">&nbsp;</div>
          Text Color : <span>#000000</span>
        </li>
        <li class="right">
          <div id="bgRed" class="red slider" data-spinner="spBgRed" data-
type="bg" ></div><input type="text"  value="255" id="spBgRed" data-
slider="bgRed" readonly="readonly"  />
          <div id="bgGreen" class="green slider" data-spinner="spBgGreen"
data-type="bg" ></div><input type="text" value="255" id="spBgGreen" data-
slider="bgGreen" readonly="readonly"   />
          <div id="bgBlue" class="blue slider" data-spinner="spBgBlue"
data-type="bg" ></div><input type="text" value="255" id="spBgBlue" data-
slider="bgBlue" readonly="readonly" />

          <div class="clear">&nbsp;</div>
          Background Color : <span>#ffffff</span>
        </li>
      </ul>
    </div>
    <script src="js/jquery-1.10.2.js"></script>
    <script src="js/jquery-ui-1.10.4.custom.min.js"></script>
    <script src="js/colorpicker.js"></script>
  </body>
</html>
```

We started by including the jQuery UI CSS file inside the `head` section. Proceeding to the `body` section, we created a `div` with the `container` class, which will act as parent div for all the page elements. Inside this div, we created another div with `id` value `textBlock` and a `ui-state-highlight class`. We then put some text content inside this `div`. For this example, we have made three paragraph elements, each having some random text inside it.

After `div#textBlock`, there is an unordered list with the `controlsContainer` class. This `ul` element has two list items inside it. First list item has the CSS class `left` applied to it and the second has CSS class `right` applied to it.

Inside `li.left`, we created three div elements. Each of these three div elements will be converted to a jQuery slider and will represent the red (R), green (G), and blue (B) color code, respectively. Next to each of these divs is an input element where the current color code will be displayed. This input will be converted to a spinner as well.

Let's look at the first slider div and the input element next to it. The `div` has `id` `txtRed` and two CSS classes `red` and `slider` applied to it. The `red` class will be used to style the slider and the `slider` class will be used in our `colorpicker.js` file. Note that this div also has two data attributes attached to it, the first is `data-spinner`, whose value is the `id` of the input element next to the `slider div` we have provided as `sptxtRed`, the second attribute is `data-type`, whose value is `text`. The purpose of the `data-type` attribute is to let us know whether this slider will be used for changing the text color or the background color.

Moving on to the input element next to the slider now, we have set its `id` as `sptxtRed`, which should match the value of the `data-spinner` attribute on the `slider div`. It has another attribute named `data-slider`, which contains the `id` of the slider, which it is related to. Hence, its value is `txtRed`.

Similarly, all the slider elements have been created inside `div.left` and each slider has an input next to `id`. The `data-type` attribute will have the `text` value for all sliders inside `div.left`. All input elements have also been assigned a value of `0` as the initial text color will be black.

The same pattern that has been followed for elements inside `div.left` is also followed for elements inside `div.right`. The only difference is that the `data-type` value will be `bg` for slider divs. For all input elements, a value of `255` is set as the background color is white in the beginning.

In this manner, all the six sliders and the six input elements have been defined. Note that each element has a unique ID.

Finally, there is a `span` element inside both `div.left` and `div.right`. The hex color code will be displayed inside it. We have placed `#000000` as the default value for the text color inside the span for the text color and `#ffffff` as the default value for the background color inside the span for background color.

Lastly, we have included the jQuery source file, the jQuery UI source file, and the `colorpicker.js` file.

With the markup ready, we can now write the properties for the CSS classes that we used here.

Styling the content

To make the page presentable and structured, we need to add CSS properties for different elements. We will do this inside the `head` section. Go to the `head` section in the `index.html` file and write these CSS properties for different elements:

```
<style type="text/css">
  body{
    color:#025c7f;
    font-family:Georgia,arial,verdana;
    width:700px;
    margin:0 auto;
  }
  .container{
    margin:0 auto;
    font-size:14px;
    position:relative;
    width:700px;
    text-align:justify;
  }
#textBlock{
    color:#000000;
    background-color: #ffffff;
  }
  .ui-state-highlight{
    padding: 10px;
    background: none;
  }
  .controlsContainer{
      border: 1px solid;
      margin: 0;
      padding: 0;
      width: 100%;
      float: left;
  }
  .controlsContainer li{
      display: inline-block;
      float: left;
      padding: 0 0 0 50px;
      width: 299px;
  }
  .controlsContainer div.ui-slider{
      margin: 15px 0 0;
      width: 200px;
      float:left;
  }
  .left{
    border-right: 1px solid;
  }
  .clear{
    clear: both;
  }

  .red .ui-slider-range{
```

```
background: #ff0000;
}
  .green .ui-slider-range{
background: #00ff00;
}
  .blue .ui-slider-range{
background: #0000ff;
}

  .ui-spinner{
      height: 20px;
      line-height: 1px;
      margin: 11px 0 0 15px;
    }
  input[type=text]{
    margin-top: 0;
    width: 30px;
  }
</style>
```

First, we defined some general rules for page `body` and `div .container`. Then, we defined the initial text color and background color for the `div` with `id textBlock`.

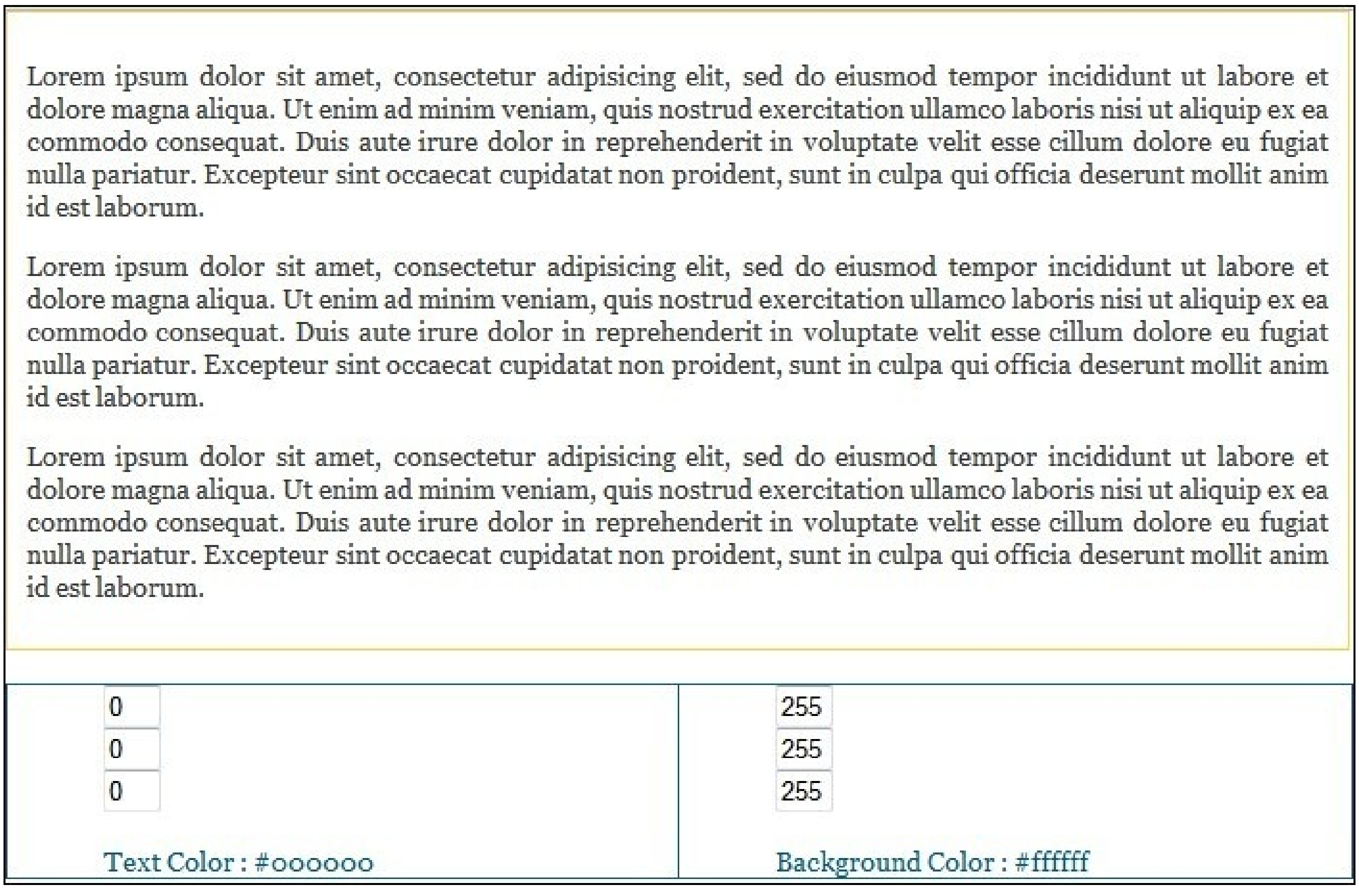
Next, we defined the CSS properties for the unordered list `ul .controlsContainer` and its list items. We have provided some padding and width to each list item.

We have also specified the width and other properties for the slider as well. Since the class `ui-slider` is added by jQuery UI to a slider element after it is initialized, we have added our properties in the `.controlsContainer div .ui-slider` rule.

To make the sliders attractive, we then defined the background colors for each of the slider bars by defining color codes for `red`, `green`, and `blue` classes.

Lastly, CSS rules have been defined for the spinner and the input box.

We can now check our progress by opening the `index.html` page in our browser. Loading it will display a page that resembles the following screenshot:



Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

0	255
0	255
0	255
Text Color : #000000	Background Color : #ffffff

It is obvious that sliders and spinners will not be displayed here. This is because we have not written the JavaScript code required to initialize those widgets. Our next section will take care of them.

Implementing the color picker

In order to implement the required functionality, we first need to initialize the sliders and spinners. Whenever a slider is changed, we need to update its corresponding spinner as well, and conversely if someone changes the value of the spinner, we need to update the slider to the correct value. In case any of the value changes, we will then recalculate the current color and update the text or background color depending on the context.

Defining the object structure

We will organize our code using the object literal pattern as we have done in earlier chapters. We will define an `init` method, which will be the entry point. All event handlers will also be applied inside this method.

To begin with, go to the `js` folder and open the `colorpicker.js` file for editing. In this file, write the code that will define the object structure and a call to it:

```
var colorPicker = {
  init : function ()
  {

  },
  setColor : function(slider, value)
  {
  },
  getHexColor : function(sliderType)
  {
  },
  convertToHex : function (val)
  {
  }
}

$(function() {
  colorPicker.init();
});
```

An object named `colorPicker` has been defined with four methods. Let's see what all these methods will do:

- `init`: This method will be the entry point where we will initialize all components and add any event handlers that are required.
- `setColor`: This method will be the main method that will take care of updating the text and background colors. It will also update the value of the spinner whenever the slider moves. This method has two parameters; the slider that was moved and its current value.
- `getHexColor`: This method will be called from within `setColor` and it will return the hex code based on the RGB values in the spinners. It takes a `sliderType` parameter based on which we will decide which color has to be changed; that is, text color or background color. The actual hex code will be calculated by the next method.
- `convertToHex`: This method will convert an RGB value for color into its corresponding hex value and return it to get a `HexColor` method.

This was an overview of the methods we are going to use. Now we will implement these methods one by one, and you will understand them in detail.

After the object definition, there is the jQuery's `$(document).ready()` event handler that will call the `init` method of our object.

The init method

In the `init` method, we will initialize the sliders and the spinners and set the default values for them as well. Write the following code for the `init` method in the `colorpicker.js` file:

```
  init : function ()
{
  var t = this;
  $( ".slider" ).slider(
  {
    range: "min",
    max: 255,
    slide : function (event, ui)
    {
      t.setColor($(this), ui.value);
    },
    change : function (event, ui)
    {
      t.setColor($(this), ui.value);
    }
  });

  $('input').spinner(
  {
    min :0,
    max : 255,
    spin : function (event, ui)
    {
      var sliderRef = $(this).data('slider');
      $('#' + sliderRef).slider("value", ui.value);
    }
  });


  $( "#txtRed, #txtGreen, #txtBlue" ).slider('value', 0);
  $( "#bgRed, #bgGreen, #bgBlue" ).slider('value', 255);
}
```

In the first line, we stored the current scope value, `this`, in a local variable named `t`.

Next, we will initialize the sliders. Since we have used the CSS class slider on each slider, we can simply use the `.slider` selector to select all of them. During initialization, we provide four options for sliders: `range`, `max`, `slide`, and `change`. Note the value for `max`, which has been set to `255`. Since the value for R, G, or B can be only between 0 and 255, we have set `max` as `255`. We do not need to specify `min` as it is 0 by default.

The `slide` method has also been defined, which is invoked every time the slider handle moves. The call back for `slide` is calling the `setColor` method with an instance of the current slider and the value of the current slider. The `setColor` method will be explained in the next section.

Besides `slide`, the `change` method is also defined, which also calls the `setColor` method with an instance of the current slider and its value.

Note

We use both the `slide` and `change` methods. This is because a `change` is called once the user has stopped sliding the slider handle and the slider value has changed. Contrary to this, the `slide` method is called each time the user drags the slider handle. Since we want to change colors while sliding as well, we have defined the `slide` as well as `change` methods.

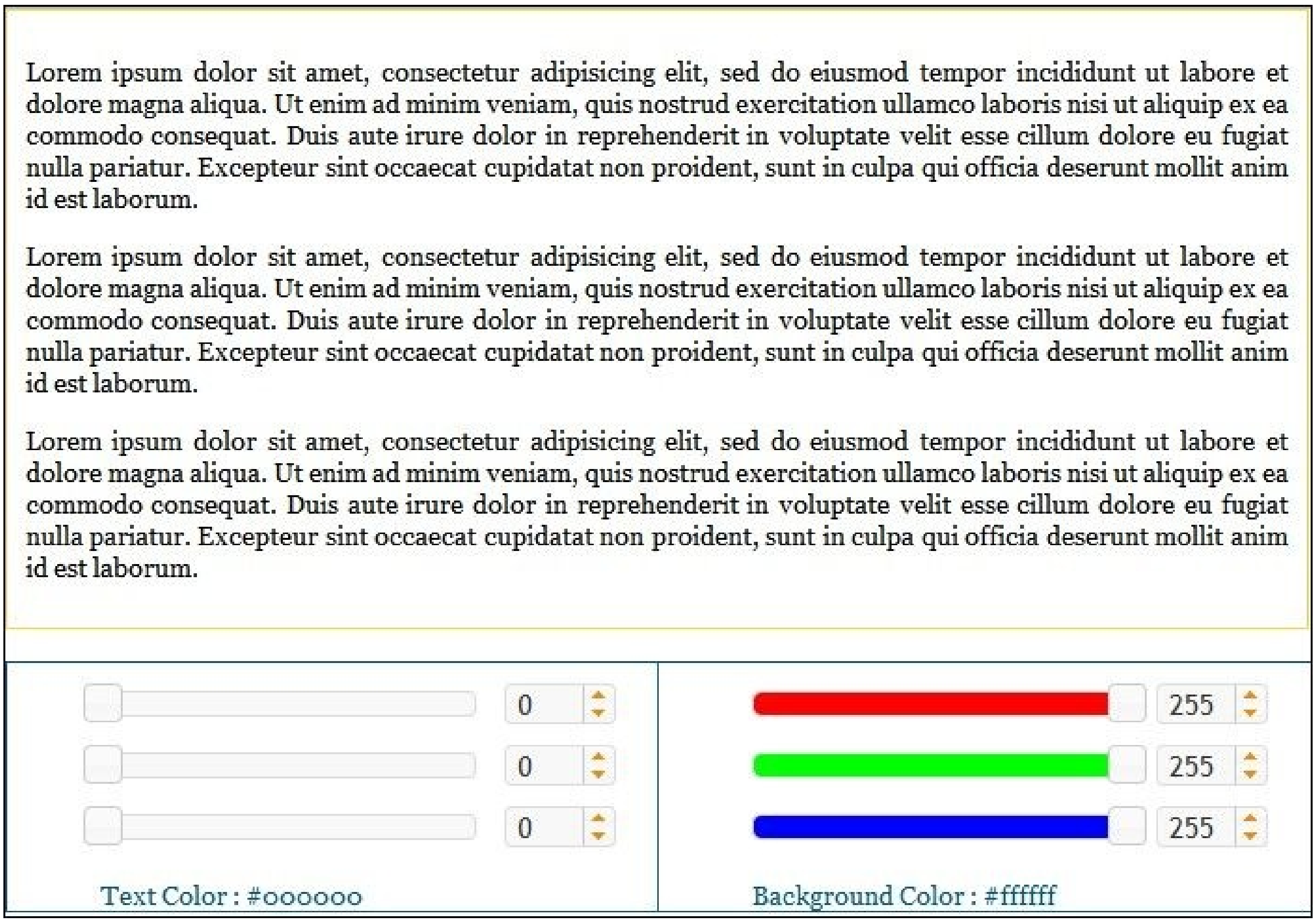
It is time to initialize the spinners now. The spinner widget is initialized with three properties. These are `min` and `max`, and the `spin`. `min` and `max` method has been set to `0` and `255`, respectively. Every time the up/down button on the spinner is clicked or the up/down arrow key is used, the `spin` method will be called. Inside this method, `$(this)` refers to the current spinner. We find our related slider to this spinner by reading the `data-slider` attribute of this spinner. Once we get the exact slider, we set its value using the value method on the slider widget.

Note

Note that calling the value method will invoke the change method of the slider as well. This is the primary reason we have defined a callback for the change event while initializing the sliders.

Lastly, we will set the default values for the sliders. For sliders inside `div.left`, we have set the value as `0` and for sliders inside `div.right`, the value is set to `255`.

You can now check the page on your browser. You will find that the slider and the spinner elements are initialized now, with the values we specified:


Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

You can also see that changing the spinner value using either the mouse or the keyboard will update the value of the slider as well. However, changing the slider value will not update the spinner. We will handle this in the next section where we will change colors as well.

Changing colors and updating the spinner

The `setColor` method is called each time the slider or the spinner value changes. We will now define this method to change the color based on whether the slider's or spinner's value was changed. Go to the `setColor` method declaration and write the following code:

```
setColor : function(slider, value)
{
  var t = this;
  var spinnerRef = slider.data('spinner');
  $('#' + spinnerRef).spinner("value", value);

  var sliderType = slider.data('type')

  var hexColor = t.getHexColor(sliderType);
  if(sliderType == 'text')
  {
      $('#textBlock').css({'color' : hexColor});
      $('.left span:last').text(hexColor);
  }
  else
  {
      $('#textBlock').css({'background-color' : hexColor});
      $('.right span:last').text(hexColor);
  }
}
```

In the preceding code, we receive the current slider and its value as a parameter. First we get the related spinner to this slider using the data attribute spinner. Then we set the value of the spinner to the current value of the slider.

Now we find out the type of slider for which `setColor` is being called and store it in the `sliderType` variable. The value for `sliderType` will either be `text`, in case of sliders inside `div.left`, or `bg`, in case of sliders inside `div.right`. In the next line, we will call the `getHexColor` method and pass the `sliderType` variable as its argument. The `getHexColor` method will return the hex color code for the selected color.

Next, based on the `sliderType` value, we set the color of `div#textBlock`. If the `sliderType` is `text`, we set the `color` CSS property of `div#textBlock` and display the selected hex code in the `span` inside `div.left`. If the `sliderType` value is `bg`, we set the background color for `div#textBlock` and display the hex code for the background color in the `span` inside `div.right`.

The getHexColor method

In the preceding section, we called the `getHexColor` method with the `sliderType` argument. Let's define it first, and then we will go through it in detail. Write the following code to define the `getHexColor` method:

```
getHexColor : function(sliderType)
{
  var t = this;
  var allInputs;
  var hexCode = '#';
  if(sliderType == 'text')
  {
    //text color
    allInputs = $('.left').find('input[type=text]');
  }
  else
  {
    //background color
    allInputs = $('.right').find('input[type=text]');
  }
  allInputs.each(function (index, element) {
    hexCode+= t.convertToHex($(element).val());
  });

  return hexCode;
}
```

The local variable `t` has stored `this` to point to the current scope. Another variable `allInputs` is declared, and lastly a variable to store the hex code has been declared, whose value has been set to # initially.

Next comes the `if` condition, which checks the value of parameter `sliderType`. If the value of `sliderType` is text, it means we need to get all the spinner values to change the text color. Hence, we use jQuery's `find` selector to retrieve all `input` boxes inside `div.left`. If the value of `sliderType` is `bg`, it means we need to change the background color. Therefore, the `else` block will be executed and all input boxes inside `div.right` will be retrieved.

To convert the color to hex, individual values for red, green, and blue will have to be converted to hex and then concatenated to get the full color code. Therefore, we iterate in inputs using the `.each` method. Another method `convertToHex` is called, which converts the value of a single input to hex. Inside the `each` method, we keep concatenating the hex value of the R, G, and B components to a variable `hexCode`. Once all iterations are done, we return the `hexCode` to the parent function where it is used.

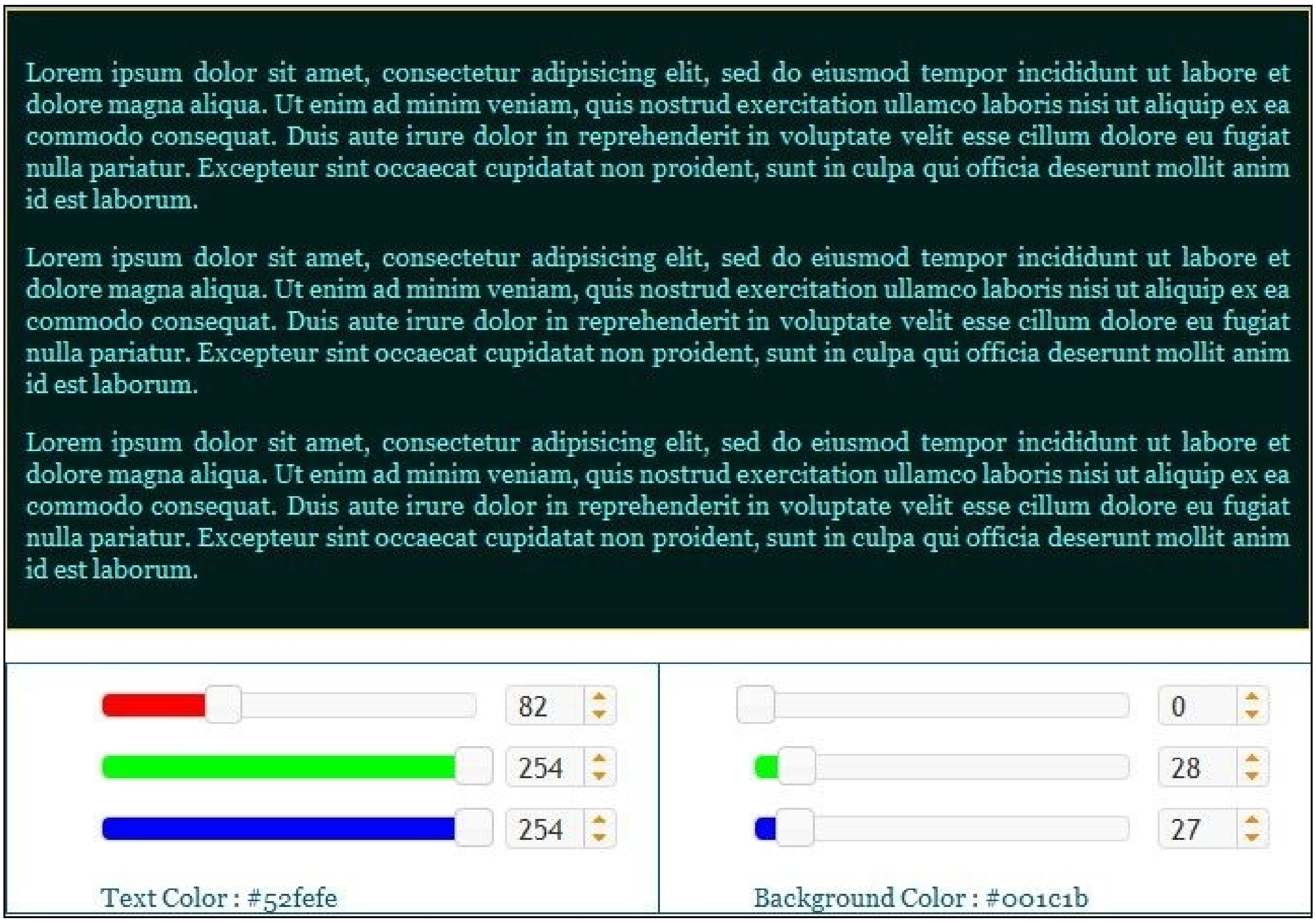
Converting to hex

`convertToHex` is a small method that accepts a value and converts it to the hex equivalent. Here is the definition of the `convertToHex` method:

```
convertToHex : function (val)
{
  var x  = parseInt(val, 10).toString(16);
  return x.length == 1 ? "0" + x : x;
}
```

Inside the method, firstly we will convert the received value to an integer using the `parseInt` method and then we'll use JavaScript's `toString` method to convert it to hex, which has base 16. In the next line, we will check the length of the converted hex value. Since we want the 6-character dash notation for color (such as `#ff00ff`), we need two characters each for red, green, and blue. Hence, we check the length of the created hex value. If it is only one character, we append a 0 to the beginning to make it two characters. The hex value is then returned to the parent function.

With this, our implementation is complete and we can check it on a browser. Load the page in your browser and play with the sliders and spinners. You will see the text or background color changing, based on their value:



Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

82
254
254

Text Color : #52fefe

0
28
27

Background Color : #001c1b

You will also see the hex code displayed below the sliders. Also note that changing the sliders will change the value of the corresponding spinner and vice versa.

Improving the Colorpicker

This was a very basic tool that we built. You can add many more features to it and enhance its functionality. Here are some ideas to get you started:

- Convert it into a widget where all the required DOM for sliders and spinners is created dynamically
- Instead of two sliders, incorporate the text and background changing ability into a single slider with two handles, but keep two spinners as usual

Summary

In our penultimate chapter, we created a basic color picker/changer using sliders and spinners. You can use it to view and change the colors of your pages dynamically.

In the last and final chapter, we will create a dashboard where we will use the knowledge of jQuery UI that we’ve covered so far. We will use multiple jQuery UI widgets and create our own as well.

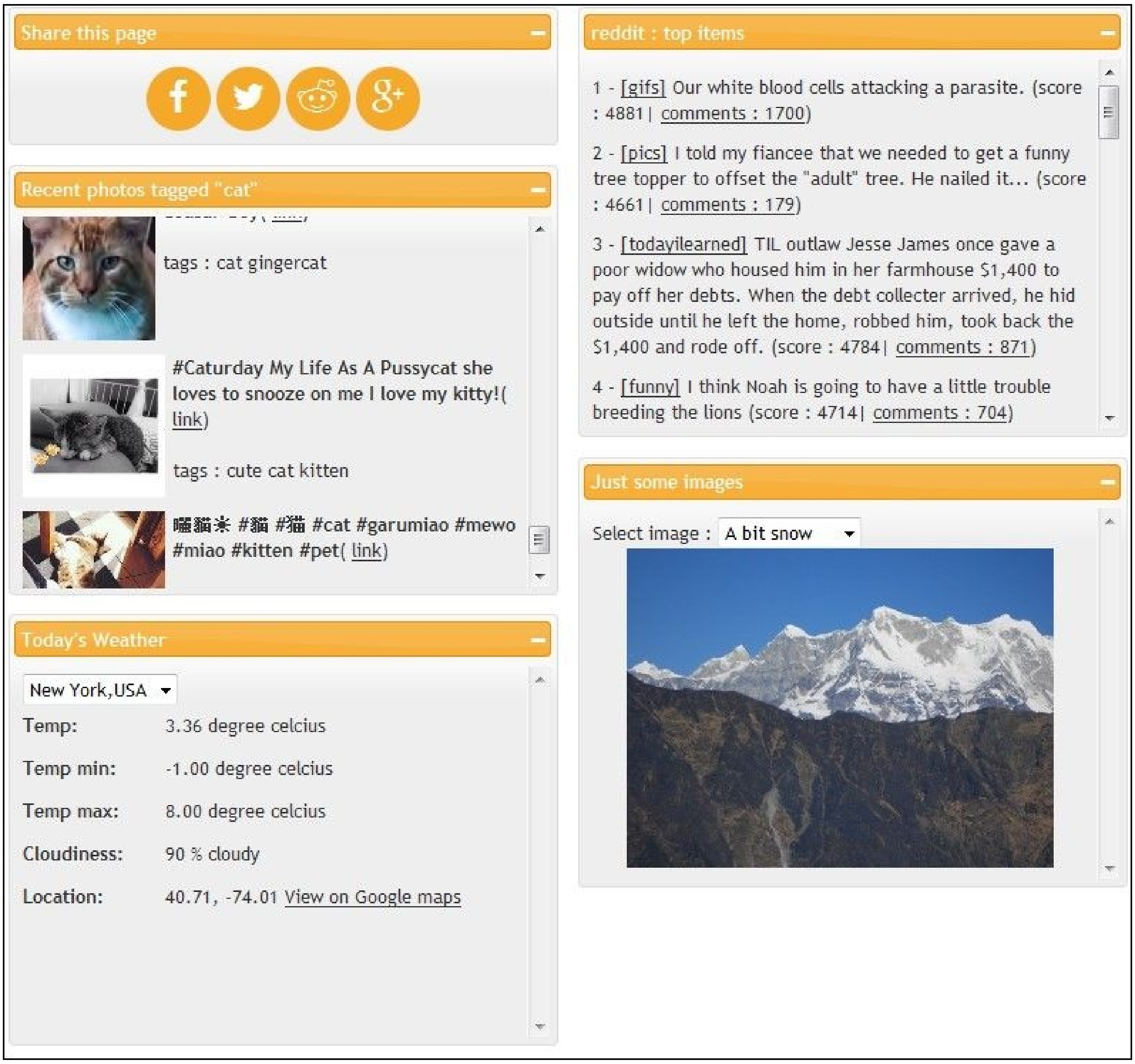
Chapter 11. Creating a Fully Functional Dashboard

We have come a long way since the start of this book and this is the final step of our journey. We have covered almost all the jQuery UI components in previous chapters. In this chapter, we will relax and create a simple dashboard with five portlets. Each of these portlets will have a widget inside them and we will use different APIs to display content inside each of them. The five portlets will be sortable as well, and you will be able to change their positions by dragging them.

The portlets that we will create are as follows:

- **Share buttons**: We will place sharing buttons for various social media sites. Clicking any of these buttons will share the page on that social media site.
- **Displaying pictures using the Flickr API**: We will search the Flickr API using a tag name and display image thumbnails and their titles. Clicking on a thumbnail will show the image in a dialog box.
- **Weather widget**: This is a widget where we will display a dropdown of places around the world and use the `Openweathermap` API to get temperature, sunrise, sunset, and coordinates of that place. Clicking on the coordinates will open the locations on Google Maps.
- **Reddit**: This widget displays latest posts from the reddit front page. We will show the post score and number of comments and link it to reddit.
- **Images**: The last widget will have a dropdown with some image names. Selecting a name from dropdown will display the image thumbnail inline. Clicking on the thumbnail will display the full size image in a modal dialog.

After we are done, the page will look like the following:



Share this page

Recent photos tagged "cat"

tags : cat gingercat

#Caturday My Life As A Pussycat she loves to snooze on me I love my kitty!(link)

tags : cute cat kitten

曬貓米 #貓 #猫 #cat #garumiao #mewo #miao #kitten #pet(link)

Today's Weather

New York,USA

Temp: 3.36 degree celcius

Temp min: -1.00 degree celcius

Temp max: 8.00 degree celcius

Cloudiness: 90 % cloudy

Location: 40.71, -74.01 View on Google maps

reddit : top items

1 - [gifs] Our white blood cells attacking a parasite. (score : 4881 | comments : 1700)

2 - [pics] I told my fiancee that we needed to get a funny tree topper to offset the "adult" tree. He nailed it... (score : 4661 | comments : 179)

3 - [todayilearned] TIL outlaw Jesse James once gave a poor widow who housed him in her farmhouse $1,400 to pay off her debts. When the debt collecter arrived, he hid outside until he left the home, robbed him, took back the $1,400 and rode off. (score : 4784 | comments : 871)

4 - [funny] I think Noah is going to have a little trouble breeding the lions (score : 4714 | comments : 704)

Just some images

Select image : A bit snow

Setting up the stage

The first step is to create the required folder structure:

1. As we have in previous chapters, create a folder named `Chapter11` inside the `MasteringjQueryUI` folder.
2. Directly inside this folder, create the HTML file named `index.html` which will contain our HTML markup.
3. Also, copy the `js` and `css` folders inside the `Chapter11` folder.
4. Now go inside the `js` folder and create a new file and name it `dashboard.js`. This file will contain all the code required to create our news reader.

Once this setup is complete, we can move to the next step, which is designing the page.

Designing the page

For the page design, we will take inspiration from the portlet script on the jQuery UI website. This script is a jQuery UI sortable example from the jQuery UI demo site. The original script can be found at http://jqueryui.com/resources/demos/sortable/portlets.html. It uses a three-column layout but we will modify it to make two columns with two widgets in each column and then create a widget in each of the columns.

Let us begin by writing the markup that we require. Open the `index.html` file in your favorite editor and write the following markup to create the page skeleton:

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Dashboard</title>
    <link href="//maxcdn.bootstrapcdn.com/font-awesome/4.2.0/css/font-
awesome.min.css" rel="stylesheet">
    <link rel="stylesheet" href="css/ui-lightness/jquery-ui-
1.10.4.custom.min.css">
  </head>
  <body>
    <div class="column">
<div class="portlet">
        <div class="portlet-header">Share this page</div>
        <div class="shareBox portlet-content">
          <a href="#" type="fb">
            <i class="fa fa-facebook"></i>
          </a>
          <a href="#" type="tweet">
            <i class="fa fa-twitter"></i>
          </a>
          <a href="#" type="reddit">
            <i class="fa fa-reddit"></i>
          </a>
          <a href="#" type="gplus">
            <i class="fa fa-google-plus"></i>
          </a>
        </div>
      </div>

      <div class="portlet">
        <div class="portlet-header">Recent photos titled cats </div>
        <div class="portlet-content flickrPics">
          <ul id="pics"></ul>
        </div>
      </div>
      <div class="portlet">
        <div class="portlet-header">Today's Weather</div>
        <div class="portlet-content">
          <select id="selCity"></select>
          <p id="loadingWeather">Loading…</p>
          <div id="weatherInfo">
            <ul>
```

```
              <li>
                <label> Temp: </label>
                <span id="temp"></span>
              </li>
              <li>
                <label> Temp min: </label>
                <span id="tempMin"></span>
              </li>
              <li>
                <label> Temp max: </label>
                <span id="tempMax"></span>
              </li>
              <li>
                <label> Cloudiness: </label>
                <span id="cloudiness"></span>
              </li>
              <li>
                <label> Location: </label>
                <span id="location"></span>
              </li>
            </ul>
          </div>
        </div>
      </div>

    </div>

    <div class="column">
      <div class="portlet">
        <div class="portlet-header">reddit : top items</div>
        <div class="portlet-content" id="reddit">
        </div>
      </div>
      <div class="portlet">
        <div class="portlet-header">Just some images</div>
        <div class="portlet-content">
          Select image : <select id="imageSelector"></select>
          <div id="thumbnail"></div>
        </div>
      </div>
    </div>

    <div id="dialog"></div>

    <script src="js/jquery-1.10.2.js"></script>
    <script src="js/jquery-ui-1.10.4.custom.min.js"></script>
    <script src="js/dashboard.js"></script>
  </body>
</html>
```

Let us look at this HTML in detail. Inside the head section, we provided the page title first. After that we loaded a CSS file from the `font-awesome` CDN. If you are not familiar, font-awesome is a great utility to display scalable vector icons on your web pages. Currently, it boasts of more than 470 different icons. We are using it to display the social media icons.

Next we proceed by linking the jQuery UI CSS file inside the `head` section. Inside the `body` tag, we have two divs with the CSS `class column` applied to them. Among these, the first has three divs inside it and second has two more divs inside it. Each of the inner divs had class `portlet` applied to them. This makes a total of four divs with `class portlet`.

Now each of these `portlet` divs has two more divs inside them. One is for portlet header and the other is to display the content of the portlet. The first `div` has the `portlet-header class` and the second `div` has the `portlet-content class` applied to it.

The first portlet has the **Share this page** title. Its content `div` has another CSS class named `shareBox` applied to it. Inside the `shareBox div`, we placed four hyperlinks and each one has an attribute called `type` so that we may identify them later using jQuery. The value for `type` attribute assigned to these hyperlinks is `fb`, `tweet`, `reddit` and `gplus` respectively.

Note

Note that we could have created the sharing links in HTML itself. But since we want to keep it generic, we will use JavaScript to make it dynamic so that the page link will be picked automatically.

Each hyperlink also has an icon inside it. These are font-awesome icons which are placed using the `<i>` tag and specific icons are displayed by applying the icon-related CSS class to it. Note that all icons must have a common class `fa`. For the icons that we are using, the class names that are required are `fa-facebook`, `fa-twitter`, `fa-reddit`, and `fa-google-plus` respectively.

Note

The full list of all the icons is available at http://fortawesome.github.io/Font-Awesome/icons/ where new icons are added periodically.

The second portlet has the title **Recent photos tagged “cat”** and its `content div` has a `ul` with `id pics` inside it. We will create the DOM inside it using the Flickr API.

The third portlet is for displaying weather. Its title is **Today’s Weather**. Inside this portlet, there is a select box where we will display the list of cities for which weather will be displayed. Next to it is a loading placeholder element, which we will display while data is being fetched from the open weather map API. Lastly, there is another list where we created different placeholders to display temperature, minimum and maximum temperature, sunrise, sunset, and the location coordinates.

The fourth portlet (or first portlet in second column) is for displaying reddit front page posts. Its title is **reddit**: top items and the portlet content div has the `id reddit`. This is where we will insert the DOM for reddit posts.

The fifth and last portlet has the title **Just some images**. Its `content` div has a select box inside it and another div. The select box will be populated with some image names and the div will display the thumbnail for the selected image.

After the columns, we have another div with `id dialog`. It will be converted to the jQuery UI dialog box and the images for the first and last portlet will be displayed in it.

Finally, just before the `body` tag closes, refer to the jQuery source file, the jQuery UI source file, and the currently empty `dashboard.js` file.

This prepares our page skeleton that we now need to stylize using CSS properties for different elements.

Styling the content

We will need some generic CSS rules for columns and portlets, such as their width and height. Besides this, we will need some other rules for the content inside different portlets. In the head section of `index.html`, after jQuery UI CSS file is referred, let's write the following rules now for different page elements inside `style` tag:

```
body
{
  font-family:arial,verdana;
  font-size:12px;
  margin: 0px auto;
  width: 800px;
}
.column
{
  width: 400px;
  float: left;
  padding-bottom: 100px;
}
.portlet
{
  margin: 0 1em 1em 0;
  padding: 0.3em;
}
.portlet-header
{
  padding: 0.2em 0.3em;
  margin-bottom: 0.5em;
  position: relative;
}
.portlet-toggle
{
  position: absolute;
  top: 50%;
  right: 0;
  margin-top: -8px;
  cursor:pointer;
}
.portlet-content
{
  height: 250px;
  overflow-y: scroll;
  padding: 0.4em;
}
.portlet-placeholder
{
  border: 1px dotted black;
  margin: 0 1em 1em 0;
  height: 250px;
}
.shareBox
{
  text-align:center;
  overflow:hidden !important;
  height:auto !important;
}


.shareBox a
{
  background: none repeat scroll 0 0 #f6a828;
  border-radius: 35px;
  color: #fff;
  display: inline-block;
  font-size: 25px;
  font-weight: bold;
  padding: 5px;
  text-align: center;
  width: 35px;
  height:35px;
}


ul
{
  list-style: outside none none;
  margin: 0;
  padding: 0;
}
.flickrPics li
{
  height: 100px;
  overflow: hidden;
  padding: 5px 0;
}
.flickrPics li img
{
  float: left;
  margin-right: 5px;
  max-height: 100px;
}

#reddit ul
{
  list-style: outside none none;
  margin: 0;
  padding: 0;
}
#reddit li , #weatherInfo li
{
  padding: 5px 0;
  min-height:20px;
}

#reddit li span
{
  text-decoration:underline;
}
#reddit a
```

```
{
  text-decoration:none;
}

#reddit a.comments
{
  text-decoration:underline;
}

#weatherInfo label
{
  font-weight:bold;
  width:100px;
  display:block;
  float:left;
}
#weatherInfo span
{
  float:left;
}

#thumbnail
{
  cursor:pointer;
  text-align:center;
}
```

The CSS properties above define the look and feel of elements inside various portlets. First of all, we defined the rules common for all portlets by declaring the properties for the classes `column`, `portlet`, `portlet-header`, `portlet-toggle`, `portlet-content`, and `portlet-placeholder`.

After the common CSS rules, properties specific to contents of a portlet have been written.

Do not bother checking the page in the browser yet. We still have to add some more classes to the portlet header and portlet content, which we will do with jQuery. Let us move to the next section where we start coding the page.

Getting the code structure ready

Let's begin by defining the structure of the JavaScript object that we will need to perform various tasks. Open the `js/dashboard.js` file and write the following code:

```
$(document).ready(function()
{
  var dashboard =
  {
    imageArr : [],
    init : function()
    {
      this.initPortlets();
      this.initSharing();
      this.initFlickr();
      this.initReddit();
      this.setupWeather();
      this.setupImageSelector();
    },
    initPortlets : function()
    {
    },
    initSharing : function()
    {
    },
    initFlickr : function()
    {
    },
    initReddit : function()
    {
    },
    setupWeather : function()
    {
    },
    setupImageSelector : function()
    {
    }
  }
  dashboard.init();
});
```

We start off by creating a dashboard object and defining an array named `imageArr` and six methods. The `imageArr` array will be populated when we create the fourth portlet and display images.

Among the methods, the first method is `init`, which acts as an entry method to the whole object. Inside `init`, we call the `initPortlets` method, which will style the portlets and add the `sortable` behavior. It will also make portlets toggleable.

The second method is `initSharing`, where we will bind event handlers for click event on any of the social media sharing links. On clicking any such link, a new browser window will be opened to share the link on that particular platform. Next are `init` methods for the four portlets: `initFlickr`, which will create the first portlet, that is, Flickr pictures;

`initReddit` for reddit portlet; `setUpWeather`, which will populate the third portlet for weather; and lastly, `setupImageSelector`, which will display the images dropdown in the last portlet and add events to display selected images.

After the definition on the dashboard object, we call the `dashboard.init()` method to initialize the application. This will be the first method that will be fired after the page has loaded.

Now that we are clear with all the methods and their functionalities, we can proceed to implement them. We will start by initializing the portlets first.

Initializing the portlets

To initialize the portlets, we will turn the `column` div into jQuery UI's `sortable` components. Since there are two columns, we will make the connection as well so that portlets of a column could be dragged into another column. We will also add a toggle button to portlet header.

Write the following code inside the `initPortlets` method to initialize them:

```
$(".column").sortable(
{
  connectWith: ".column",
  handle: ".portlet-header",
  cancel: ".portlet-toggle",
  placeholder: "portlet-placeholder ui-corner-all"
});

$(".portlet")
  .addClass("ui-widget ui-widget-content ui-helper-clearfix ui-corner-all")
  .find(".portlet-header")
  .addClass("ui-widget-header ui-corner-all")
  .prepend("<span class='ui-icon ui-icon-minusthick portlet-toggle'>
</span>");

$(".portlet-toggle").click(function()
{
  var icon = $(this);
  icon.toggleClass("ui-icon-minusthick ui-icon-plusthick");
  icon.closest(".portlet").find(".portlet-content").toggle('fast');
});
$('#loadingWeather').hide();
$('#weatherInfo').hide();
```

First we convert the `column` div to `sortable`, which will make the `column` div inside, the `sortable` component. We also specified four options for sortables, which are described here:

- `connectWith`: Since we have two columns and we want the `column` div inside to be able to move between those, we need to set the `connectWith` option. This is why we set its value to `.column`
- `handle`: This is the element that will be used to drag the portlet. We specified its value as `.portlet-header`.
- `cancel`: This is the selector which will not allow dragging of portlet. Since the toggle button is also in the portlet header, we allowed its CSS class the `cancel` value.
- `placeholder`: This is the class name that is applied to the empty space created due to dragging an original element. We will display a dotted border for visual effects.

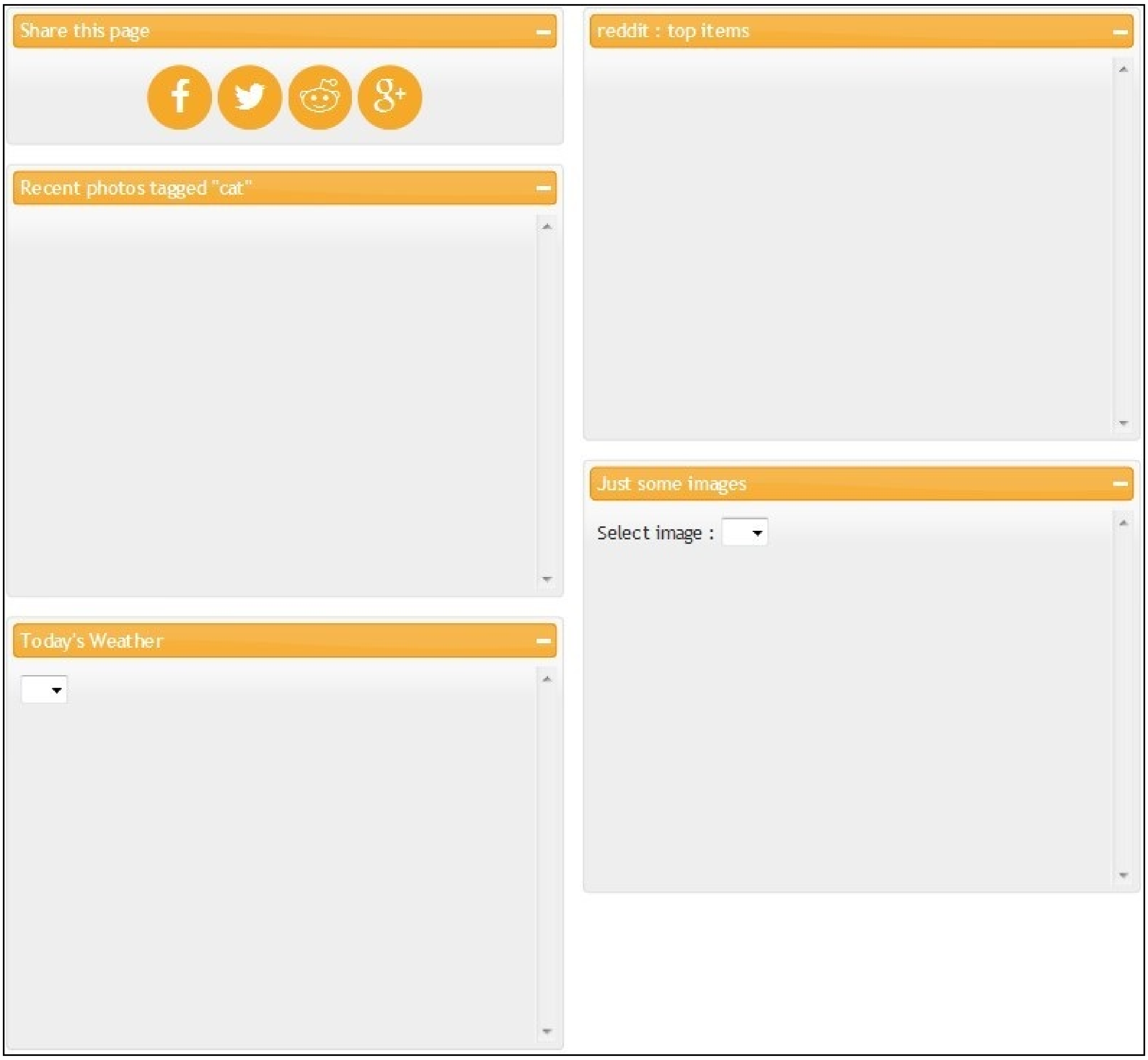
Now for each portlet, we add jQuery UI themeroller classes to the `portlet` div as well as to the `.portlet-header` div inside it. After this, we create a `span` element and `prepend` it to the `portlet-header` div. This `span` element has CSS classes `ui-icon` and `ui-icon-minusthick` from jQuery UI's theme, which will display a toggle icon in the `portlet-`

header div.

Next is the event handler for the click event of the portlet-toggle span that toggles the icon in the portlet header. It uses jQuery’s toggleClass method to alternate between the ui-icon-minusthick and ui-icon-plusthick classes to change icons. Last line toggles the visibility of the portlet-content div for that portlet.

Last two lines hide the weather loading indicator in second portlet as well as the template for weather information.

Now we can check what our hard work has produced so far. Browse to the index.html file using your web server in the browser and you will see the portlets in two columns, as displayed in the following screenshot:



The portlets are now ready to display data. We will begin by creating the first portlet, that is, implementing the social media sharing buttons.

Implementing sharing buttons

We have four sharing buttons and each of these needs to be configured differently to share. When the user will click an icon, we will open a new window and based on the icon that was clicked, redirect the user to the appropriate page. Write the following code inside the `initSharing` method that will add the event handler for the `click` event of sharing links:

```
$('.shareBox a').on('click', function()
{
  var type = $(this).prop('type');
  dashboard.sharePage(type);
});
```

Content for the first portlet has the `shareBox` class assigned to it and it has sharing links inside it. In the above code, we added the click event handler for all the links inside the `.shareBox` div. The callback method for the event handler first gets the value of type attribute for the clicked link. We then call the `sharePage` method and pass the value of the type property to it. This method will send users to the relevant social media page.

To implement this method go to the `dashboard.js` file again and add the following code for the method after the `initSharing` method definition:

```
sharePage : function(shareType)
{
  var pageUrl = encodeURIComponent(document.location);
  var shareUrl;
  switch(shareType)
  {
    case 'fb':
      shareUrl = 'https://www.facebook.com/sharer/sharer.php?u=' + pageUrl;
    break;

    case 'tweet':
      shareUrl = 'https://twitter.com/intent/tweet?text=Check out my
page&url='+ pageUrl +'&via=v08i';
    break;

    case 'reddit':
      shareUrl = 'http://www.reddit.com/submit?url=' + pageUrl;
    break;

    case 'gplus':
      shareUrl = 'https://plus.google.com/share?url=' + pageUrl;
    break;

    default :
    return false;
  }
  window.open(shareUrl , '', 'width=600,height=500');
},
```

The `sharePage` method receives `shareType` as a parameter, that is, it can be either `fb`, `reddit`, `tweet` or `gplus`. In the first line inside the method, we take the page URL using

JavaScript's `document.location` and then encode it using the `encodeURIComponent` function. The next line has declared a variable which will store the value of sharing URL.

Now based on the value of the `shareType` parameter, we implemented a `switch` case that will create the sharing URL. Facbook, Twitter, reddit, and Google Plus each have specific URL formats to share data. Described here are the URL formats of each of these sites:

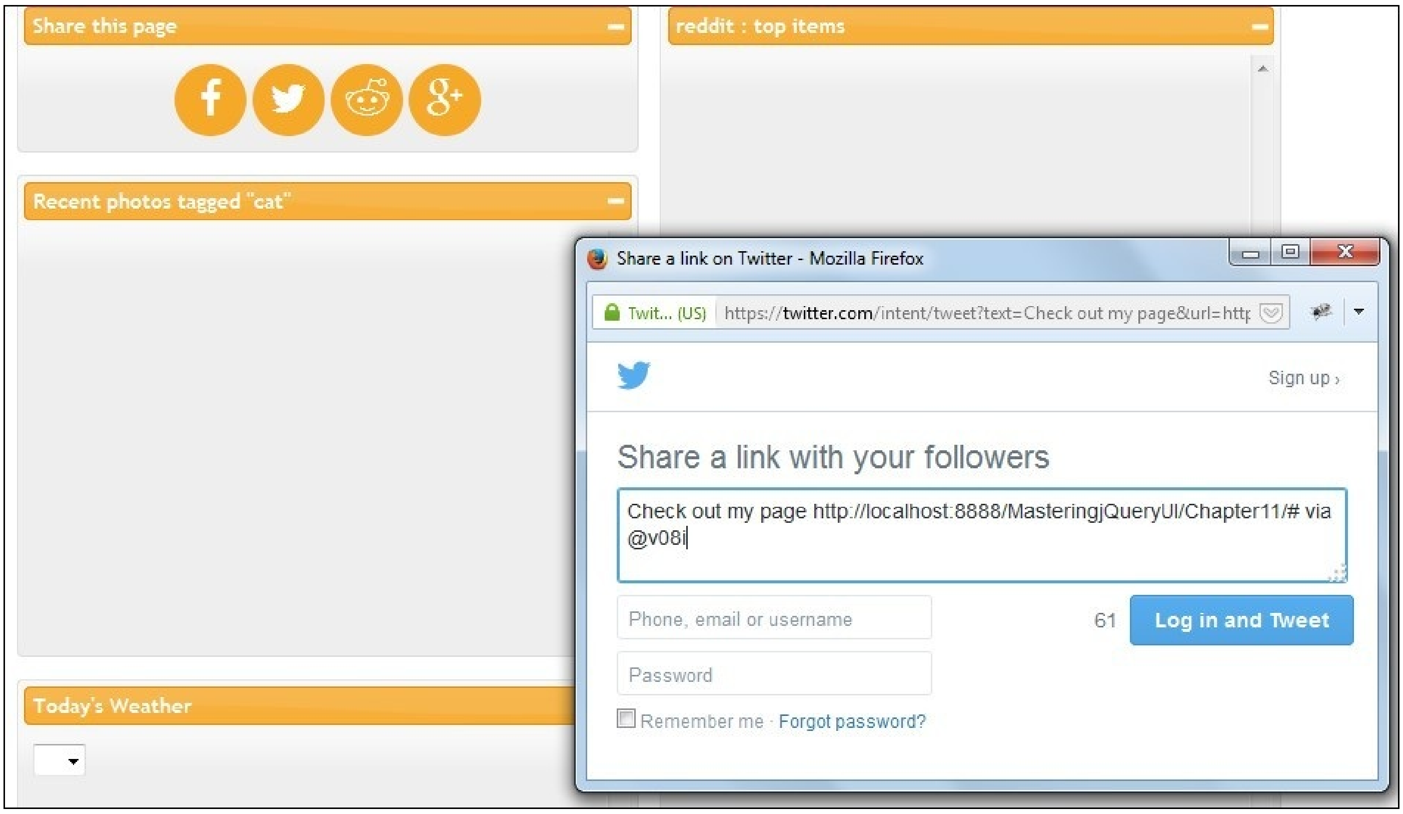
- **Facebook**: To share a URL on Facebook, the required format is https://www.facebook.com/sharer/sharer.php?u=, where the value of u in the query string is the URL you want to share.
- **Twitter**: The sharing URL for Twitter is https://twitter.com/intent/tweet?text=&url=&via=. Here text is the tweet text, URL is the URL you want to share, and value of via could be a twitter handle. via is not a mandatory field so you can skip it if not required.
- **Reddit**: To submit a URL to reddit, the required URL format is http://www.reddit.com/submit?url=, where the value of URL must be the the URL we are sharing.
- **Google Plus**: The URL format of Google Plus is similar to reddit. It has the format https://plus.google.com/share?url=.

For each of the sites in the `switch` statement, we created the URLs. Once the URL is ready, we use JavaScript's `window.open` method to open the URL in a new window. We pass three parameters to `window.open`; the first is the URL we just created, second is an empty string (the second parameter is the window name but we do not need it), and third is the width and height of the new window that will be opened. If you do not provide the third parameter, the new window will be opened in a new browser tab instead of new a pop-up window.

Note

Make sure you have popups enabled in the browser, otherwise the new window will not be visible.

We can now check our sharing feature on the page. Load the `index.html` file in your browser and click on any of the sharing links. Here is a screenshot after the Twitter sharing icon is clicked on:


Share this page
reddit : top items
Recent photos tagged "cat"
Today's Weather
Share a link on Twitter - Mozilla Firefox
Twit... (US) https://twitter.com/intent/tweet?text=Check out my page&url=http
Sign up ›
Phone, email or username
Password
Remember me · Forgot password?
61
Log in and Tweet

Share a link with your followers

Check out my page http://localhost:8888/MasteringjQueryUI/Chapter11/# via @v08i

Note that it contains the localhost URL, which is no good for anyone else. But you get the idea. It is obvious that any other URL can be shared.

Displaying Flickr photos

To search photos from Flickr, we will use the Flickr tags search API. The API is very simple to implement and provides the JSON format in response apart from XML. Since we will have to make a cross-domain request in order to contact the API, we will use JSONP to fetch the data. The Flickr API allows a callback parameter in the URL for JSONP requests. To implement the Flickr API, go to the `initFlickr` method in your file and write the following code inside it:

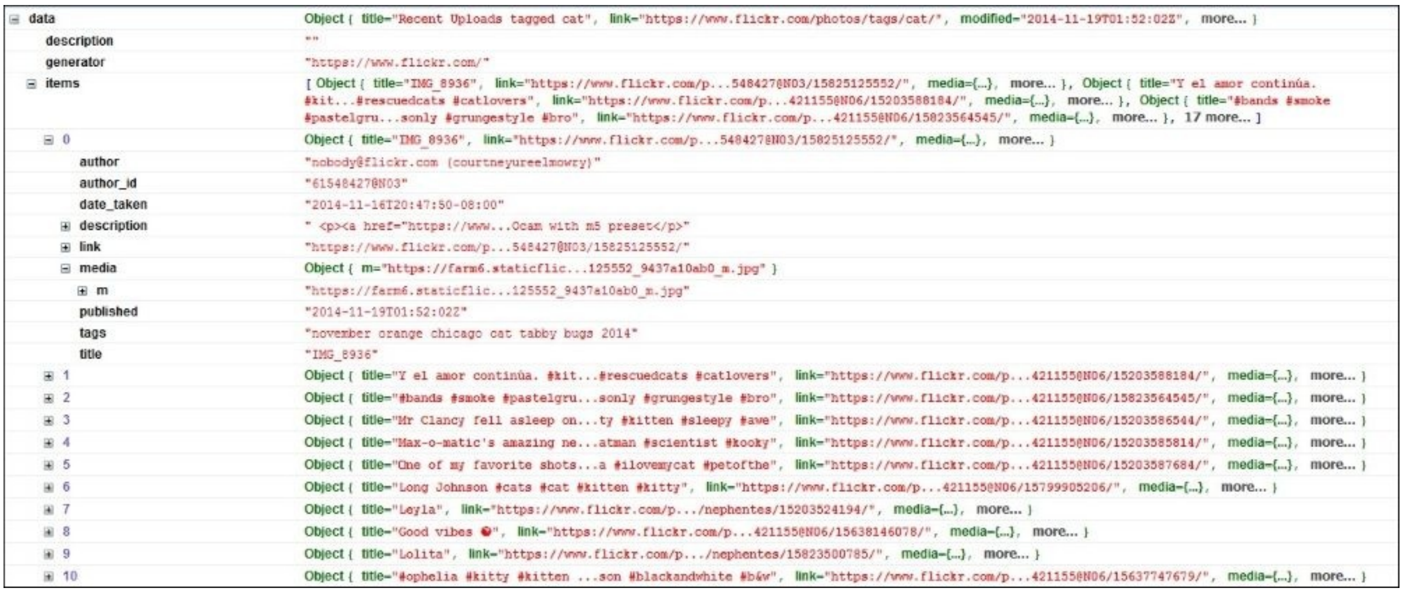
```
$.getJSON('https://api.flickr.com/services/feeds/photos_public.gne?
jsoncallback=?',
{
   tags: 'cat',
   format: 'json'
},
function(data)
{
  var str = '';
  $.each(data.items, function(i,item)
  {
    str+= '<li>';
    str+= '<a class="media" href="javascript:;" data-img="' + item.media.m
+ '">';
    str+= '<img src="' + item.media.m + '">';
    str+= '</a>';
    var permaLink = '<a href="' + item.link +'" target="_blank">link</a>';
    str+= '<strong>'+item.title+'</strong>( ' + permaLink + ')<br><br>tags
: ' + item.tags;
    str+= '</li>';
  });
  $('#pics').html(str);
});

$('#pics').on('click', 'a.media', function()
{
  var img = $(this).data('img');
  $('#dialog').html('<img src="' + img + '">').dialog({modal : true});
});
```

The first line uses jQuery's `getJSON` method to make a call to the Flickr API. We set the value of `format` as `json` in the request. The value of the `tags` parameter is set to `cat`, which means it will search for pictures tagged as "cat". The `tags` property can be any string that you want to search on Flickr.

Note

The URL that is created will be https://api.flickr.com/services/feeds/photos_public.gne?jsoncallback=&tags=cat&format=json in this case.

Once data is fetched from the service, the callback method will be invoked and it will receive `data` as a parameter. Following is the JSON response format that is received from Flickr. It will be required in order to create DOM with this response:



```
data            Object { title="Recent Uploads tagged cat", link="https://www.flickr.com/photos/tags/cat/", modified="2014-11-19T01:52:02Z", more... }
  description   ""
  generator     "https://www.flickr.com/"
  items         [ Object { title="IMG_8936", link="https://www.flickr.com/p...548427@N03/15825125552/", media={...}, more... }, Object { title="Y el amor continúa.
                #kit...#rescuedcats #catlovers", link="https://www.flickr.com/p...421155@N06/15203588184/", media={...}, more... }, Object { title="#bands #smoke
                #pastelgru...sonly #grungestyle #bro", link="https://www.flickr.com/p...421155@N06/15823564545/", media={...}, more... }, 17 more... ]
    0           Object { title="IMG_8936", link="https://www.flickr.com/p...548427@N03/15825125552/", media={...}, more... }
      author      "nobody@flickr.com (courtneyureelmowry)"
      author_id   "61548427@N03"
      date_taken  "2014-11-16T20:47:50-08:00"
      description " <p><a href="https://www...0cam with m5 preset</p>"
      link        "https://www.flickr.com/p...548427@N03/15825125552/"
      media       Object { m="https://farm6.staticflic...125552_9437a10ab0_m.jpg" }
        m         "https://farm6.staticflic...125552_9437a10ab0_m.jpg"
      published   "2014-11-19T01:52:02Z"
      tags        "november orange chicago cat tabby bugs 2014"
      title       "IMG_8936"
    1           Object { title="Y el amor continúa. #kit...#rescuedcats #catlovers", link="https://www.flickr.com/p...421155@N06/15203588184/", media={...}, more... }
    2           Object { title="#bands #smoke #pastelgru...sonly #grungestyle #bro", link="https://www.flickr.com/p...421155@N06/15823564545/", media={...}, more... }
    3           Object { title="Mr Clancy fell asleep on...ty #kitten #sleepy #awe", link="https://www.flickr.com/p...421155@N06/15203586544/", media={...}, more... }
    4           Object { title="Max-o-matic's amazing ne...atman #scientist #kooky", link="https://www.flickr.com/p...421155@N06/15203585814/", media={...}, more... }
    5           Object { title="One of my favorite shots...a #ilovemycat #petofthe", link="https://www.flickr.com/p...421155@N06/15203587684/", media={...}, more... }
    6           Object { title="Long Johnson #cats #cat #kitten #kitty", link="https://www.flickr.com/p...421155@N06/15799905206/", media={...}, more... }
    7           Object { title="Leyla", link="https://www.flickr.com/p.../nephentes/15203524194/", media={...}, more... }
    8           Object { title="Good vibes ●", link="https://www.flickr.com/p...421155@N06/15638146078/", media={...}, more... }
    9           Object { title="Lolita", link="https://www.flickr.com/p.../nephentes/15823500785/", media={...}, more... }
    10          Object { title="#ophelia #kitty #kitten ...son #blackandwhite #b&w", link="https://www.flickr.com/p...421155@N06/15637747679/", media={...}, more... }
```

As you can see, in the figure above, the images are inside the `items` array. Using jQuery's `$.each` method we loop in this array and create the DOM. There are many properties for each image but we are interested in its `title`, `image`, and `permalink`.

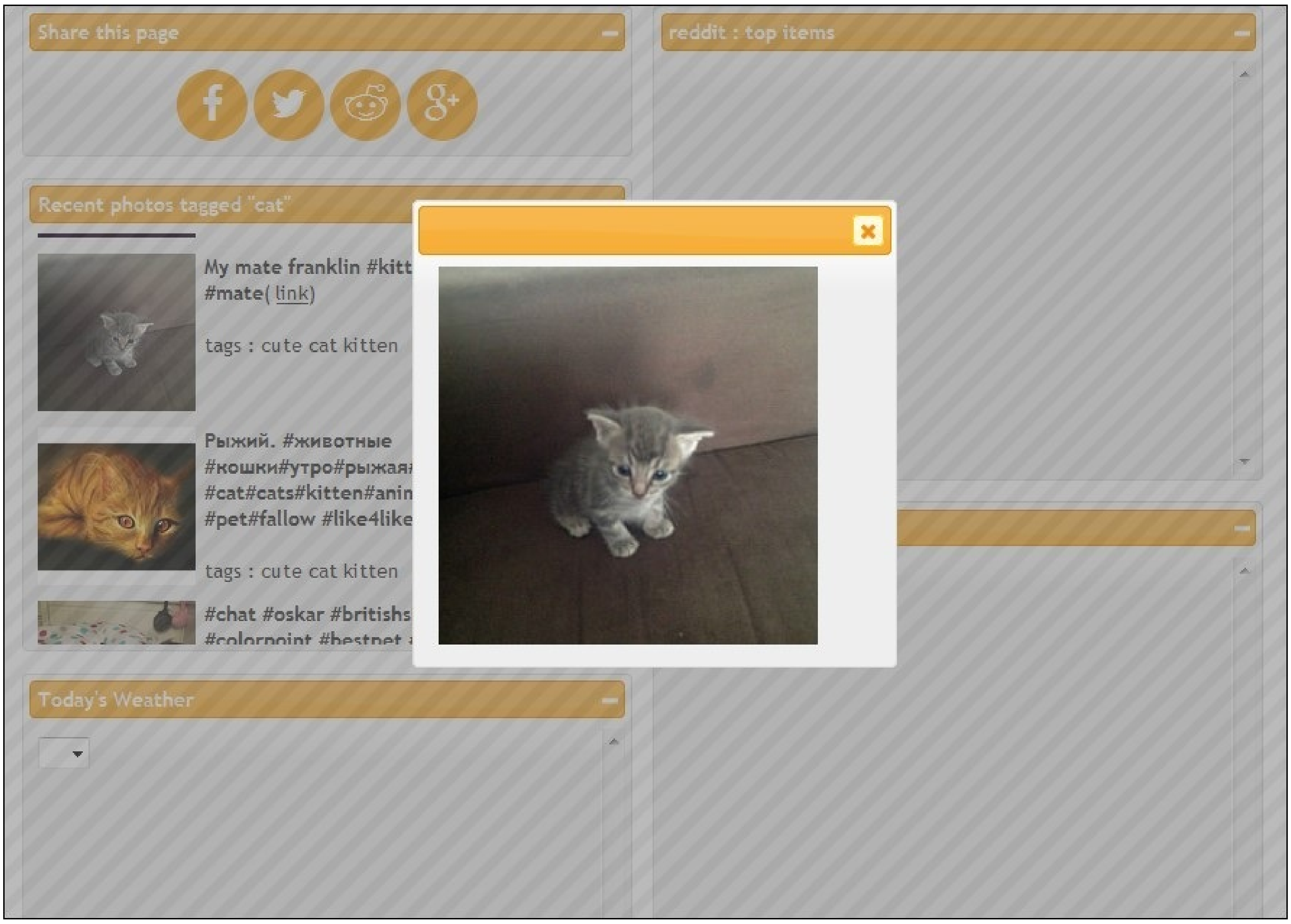
We create a `li` element for each of the images. Inside the `li`, we create an `img`, set its `src`, and wrap it in an anchor. We also set a data attribute `data-img` for image. Next to the image we create permalink for the image and then place it along with the title of the image and its tags from Flickr.

Once the loop is complete, we push the newly created HTML inside the `ul` list having the `id #pics`.

Next, we added an event handler for the `click` event of anchor, which contains the image. Note the CSS `class media`. Inside the event handler, we get the value of the data attribute `img`. Then we insert this image into the `#dialog` div and display the jQuery UI dialog.

This completes our Flickr widget and we can now check it in action. Reload the `index.html` page and the Flickr pictures will be displayed. Click on a thumbnail and it will be displayed in the dialog.

Here is how the page will look now:



My mate franklin #kitt
#mate(link)

tags : cute cat kitten

Рыжий. #животные
#кошки#утро#рыжая#
#cat#cats#kitten#anin
#pet#fallow #like4like

tags : cute cat kitten

#chat #oskar #britishs
#colorpoint #bestpet #

Creating a weather widget

For the weather widget, first we will create the dropdown with names of cities, then we will add the event handler for its change event. Write the following code inside the `setupWeather` method to create a dropdown and binding event handler:

```
var cities = ['Delhi, India', 'London,UK', 'New York,USA', 'Tokyo,Japan'];
var strCity = '<option value="0">select a city</option>';
$(cities).each(function(i, item)
{
  strCity+= '<option value="' + item + '">' + item + '</option>';
});
$('#selCity').html(strCity);

$('#selCity').change(function()
{
  var selection = $(this).val();
  if(selection == 0)
  {
    return;
  }
  dashboard.displayWeather(selection);

});
```

To create a cities dropdown, we created an array called `cities` that contains some city names around the world. Then we iterate in this array and create dropdown options with each city name and insert it into the dropdown with `id selCity`.

Next we bind the event handler for change event on the dropdown. In case a valid value is selected in dropdown, we call a method named `displayWeather` with the current dropdown value. We now need to implement the `displayWeather` method, which will fetch data from the API and display it.

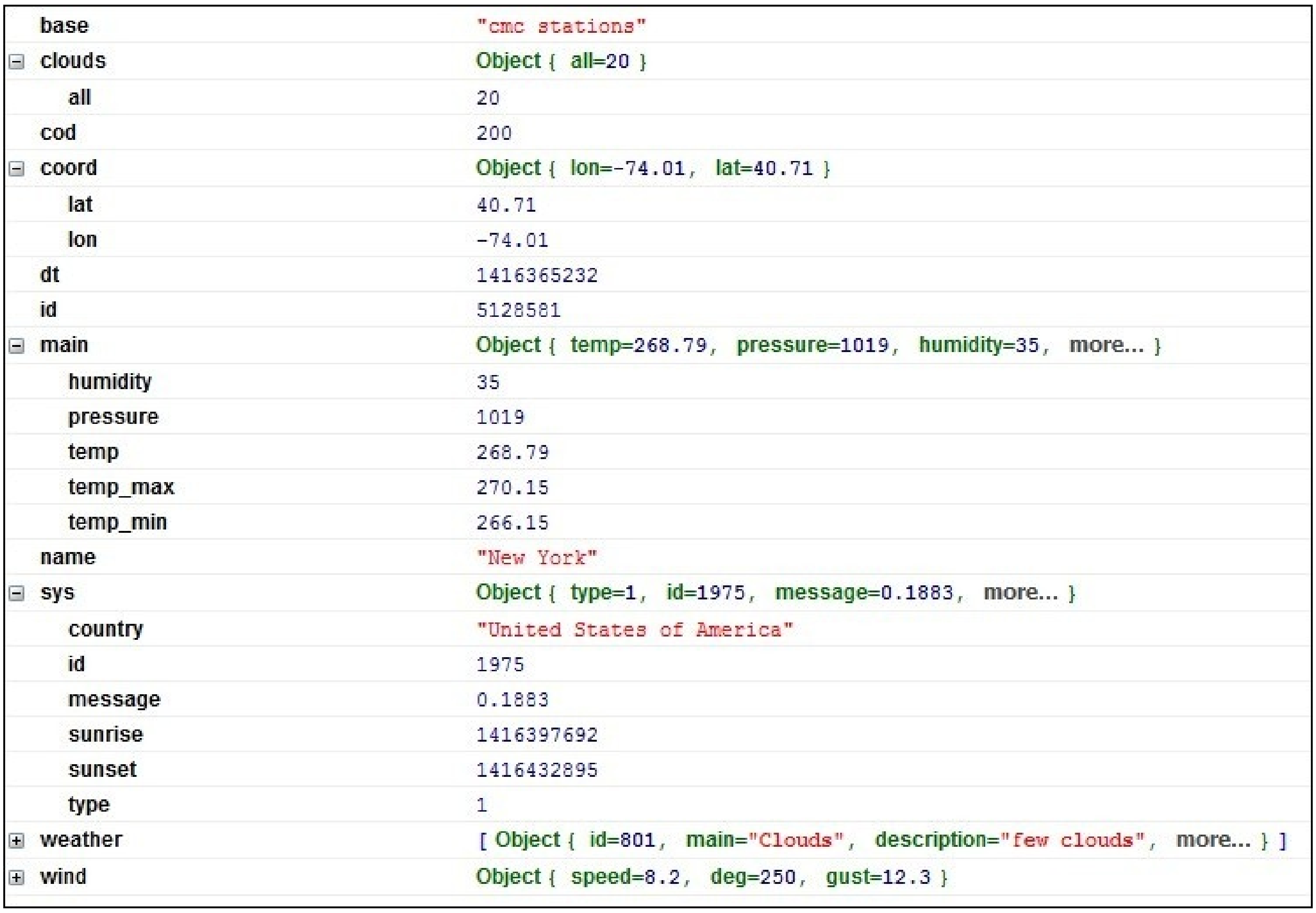
Visit the dashboard object and add a new method next to the `setupWeather` method, as shown in code below:

```
displayWeather : function(city)
{
  $('#loadingWeather').show();
  $('#weatherInfo').hide();
  var apiURL = 'http://api.openweathermap.org/data/2.5/weather?q=' + city;
  $.ajax(
  {
    url: apiURL,
    dataType: "jsonp",
    jsonp: 'callback',
    success: function(weatherData)
    {
    var x = {a : weatherData};
    console.log(x);
    $('#temp').html((weatherData.main.temp - 273.15).toFixed(2) + ' degree
celcius');
```

```
    $('#tempMin').html((weatherData.main.temp_min - 273.15).toFixed(2) + '
degree celcius');
    $('#tempMax').html((weatherData.main.temp_max - 273.15).toFixed(2) + '
degree celcius');
    $('#cloudiness').html((weatherData.clouds.all) + ' % cloudy');

    var googleUrl = 'https://www.google.com/maps?
q='+weatherData.coord.lat+',' + weatherData.coord.lon;
    var googleLink = ' <a href="' + googleUrl + '" target="_blank">View on
Google maps</a>';

    $('#location').html(weatherData.coord.lat + ', '+ weatherData.coord.lon
+ googleLink);

    $('#weatherInfo').show();
    $('#loadingWeather').hide();

    },
    error: function (a,b,c)
    {
    console.log('Error getting weather.');
    }
  });
},
```

The `displayWeather` method receives the selected city value in the `city` parameter. Inside this method, we first show you the element with ID `loadingWeather` and hide the `#weatherInfo` div.

Next we create the URL for API. The URL has format http://api.openweathermap.org/data/2.5/weather?q=, where the value of `q` is the selected city name. After creating the URL, we make an AJAX request with `dataType` set as `jsonp` and a callback value specified. In the success callback, The API response will be collected in the `weatherData` parameter, which we can now use to display different values. The response format of JSON is as follows:

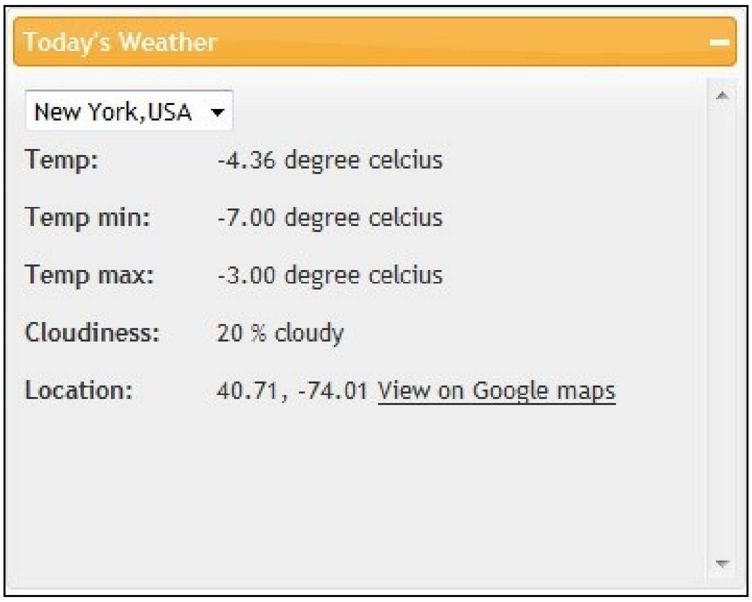


```
   base          "cmc stations"
⊟  clouds        Object { all=20 }
      all        20
   cod           200
⊟  coord         Object { lon=-74.01, lat=40.71 }
      lat        40.71
      lon        -74.01
   dt            1416365232
   id            5128581
⊟  main          Object { temp=268.79, pressure=1019, humidity=35, more... }
      humidity   35
      pressure   1019
      temp       268.79
      temp_max   270.15
      temp_min   266.15
   name          "New York"
⊟  sys           Object { type=1, id=1975, message=0.1883, more... }
      country    "United States of America"
      id         1975
      message    0.1883
      sunrise    1416397692
      sunset     1416432895
      type       1
⊞  weather       [ Object { id=801, main="Clouds", description="few clouds", more... } ]
⊞  wind          Object { speed=8.2, deg=250, gust=12.3 }
```

The temperature data is available in the `weatherData.main` object, the cloudiness data is available in `weatherData.clouds` and the coordinates of the city are available in the `weatherData.coord` object. Using these objects, we display the four elements `temp`, `tempMin`, `tempMax`, and `cloudiness` with values `temp`, `temp_min`, `temp_max`, and `clouds.all` respectively. Note that the temperature information from the API is received in Kelvin. To convert it to Celsius, we subtracted 273.15 from the values.

Next we create a link to Google Maps for the city coordinates available in the `coord` object. To display the latitude and longitudes in Google Maps, the URL must have the format https://www.google.com/maps?q=latitude,longitude. We fill the latitude and longitude values from the API response as well as display the values and the link inside the `span #location`.

The weather widget is ready now and you can check it on the page. Reload the `index.html` page and select a city name from dropdown. You will see the response as displayed in the following picture:



Today's Weather

New York,USA

Temp:	-4.36 degree celcius
Temp min:	-7.00 degree celcius
Temp max:	-3.00 degree celcius
Cloudiness:	20 % cloudy
Location:	40.71, -74.01 View on Google maps

Clicking on the **View on Google maps** link will open a new tab focused on the city.

Displaying posts from the reddit front page

We already created an example using the reddit API in Chapter 4, *Creating a Tabbed News Reader*. If you have gone through it, you can directly skip the theory section and directly use the code and implement it.

Unlike Chapter 4, *Creating a Tabbed News Reader*, in this widget we will display the posts from the reddit front page. Like earlier widgets, we will call the reddit API for JSON of front page posts. Once data is received, we will iterate in response items and create the DOM. Write the following code inside `initReddit` method to call the API:

```
var apiURL = 'http://www.reddit.com/r/all.json';
$.ajax(
{
  url: apiURL,
  dataType: "jsonp",
  jsonp: 'jsonp',
  success: function(data)
  {
  var x = {a : data};
  console.log(x);
  $('#reddit').html(dashboard.getRedditThreadList(data.data.children));
  },
  error: function (a,b,c)
  {
  alert('Error getting data');
  }
});
```

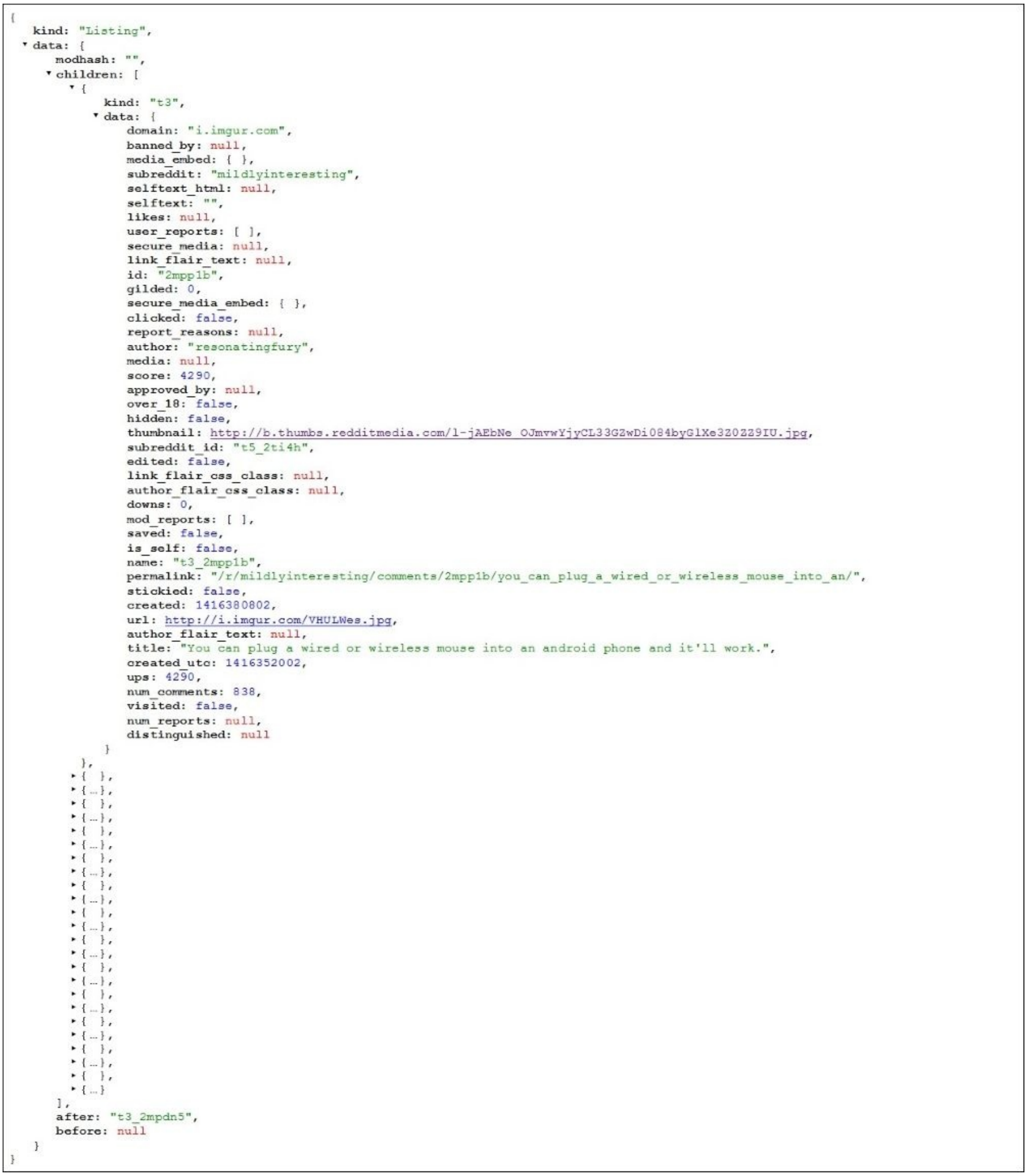
You can get the JSON for any subreddit by adding `.json` at the end of its name. Since we want to get all front page posts, we have used `all.json`. Hence the URL becomes http://www.reddit.com/r/all.json. Since it is a cross domain request we need to make it `jsonp`. Reddit supports `jsonp` and expects a callback parameter in case of `jsonp` requests, which we have provided as well as `jsonp`.

Once the request succeeds, the response is received in variable `data`. The response has another object named `data` inside it which had yet another array named `children`. This array is the actual list of posts. We pass the `children` array to another method `getRedditThreadList` which is responsible for creating the DOM.

Let us define the `getRedditThreadList` method now to create the DOM. Inside the dashboard object, write the following code to create the method and the DOM creation code inside it:

```
getRedditThreadList : function(postListing)
{
  var strHtml = '<ul>';
  for(var i = 0; i < postListing.length; i++)
  {
    var aPost = postListing[i].data;
    var permalink = 'http://reddit.com' + aPost.permalink;

    strHtml+= '<li>';
    strHtml+= (i+1) + ' - <span>[' + aPost.subreddit + ']</span> <a
href="'+aPost.url+'" target="_blank">' + aPost.title + '</a> (score : ' +
aPost.score + '| <a class="comments" href="' + permalink + '"
target="_blank"> comments : ' + aPost.num_comments + '</a>)';
    strHtml+= '</li>';
  }
  strHtml+= '</ul>';
  return strHtml;
},
```

The method `getRedditThreadList` receives the reddit response in the `postListing` parameter. Before creating DOM, let’s look at the format of an individual children node:



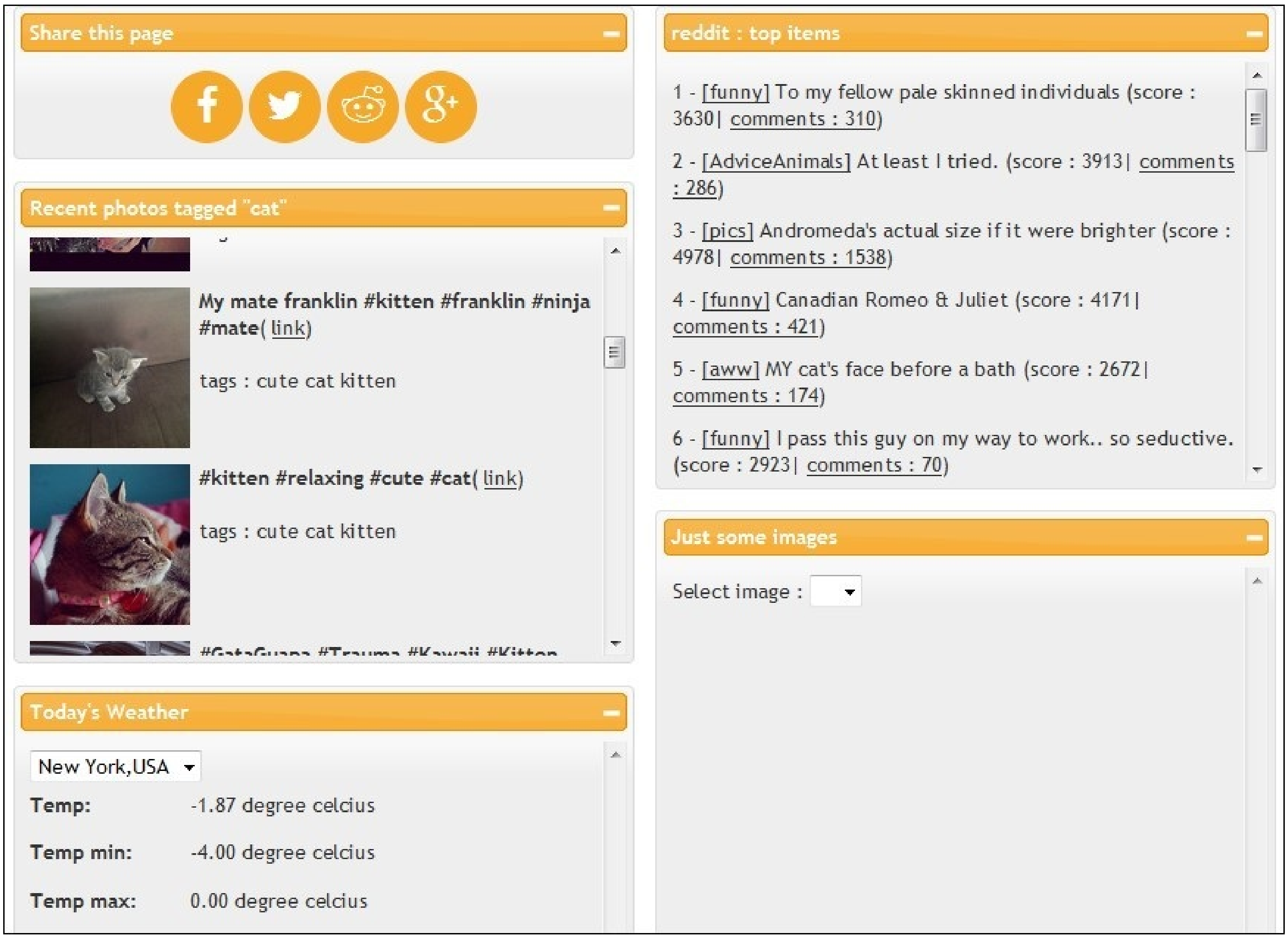
```
{
   kind: "Listing",
 ▾ data: {
       modhash: "",
     ▾ children: [
         ▾ {
               kind: "t3",
             ▾ data: {
                   domain: "i.imgur.com",
                   banned_by: null,
                   media_embed: { },
                   subreddit: "mildlyinteresting",
                   selftext_html: null,
                   selftext: "",
                   likes: null,
                   user_reports: [ ],
                   secure_media: null,
                   link_flair_text: null,
                   id: "2mpp1b",
                   gilded: 0,
                   secure_media_embed: { },
                   clicked: false,
                   report_reasons: null,
                   author: "resonatingfury",
                   media: null,
                   score: 4290,
                   approved_by: null,
                   over_18: false,
                   hidden: false,
                   thumbnail: http://b.thumbs.redditmedia.com/l-jAEbNe_OJmvwYjyCL33G2wDi084byGlXe320Z29IU.jpg,
                   subreddit_id: "t5_2ti4h",
                   edited: false,
                   link_flair_css_class: null,
                   author_flair_css_class: null,
                   downs: 0,
                   mod_reports: [ ],
                   saved: false,
                   is_self: false,
                   name: "t3_2mpp1b",
                   permalink: "/r/mildlyinteresting/comments/2mpp1b/you_can_plug_a_wired_or_wireless_mouse_into_an/",
                   stickied: false,
                   created: 1416380802,
                   url: http://i.imgur.com/VHULWes.jpg,
                   author_flair_text: null,
                   title: "You can plug a wired or wireless mouse into an android phone and it'll work.",
                   created_utc: 1416352002,
                   ups: 4290,
                   num_comments: 838,
                   visited: false,
                   num_reports: null,
                   distinguished: null
               }
           },
         ▸ { },
         ▸ {…},
         ▸ { },
         ▸ {…},
         ▸ { },
         ▸ {…},
         ▸ { },
         ▸ {…},
         ▸ { },
         ▸ {…},
         ▸ { },
         ▸ {…},
         ▸ { },
         ▸ {…},
         ▸ { },
         ▸ {…},
         ▸ { },
         ▸ {…},
         ▸ { },
         ▸ {…},
         ▸ { },
         ▸ {…},
         ▸ { },
         ▸ {…}
       ],
       after: "t3_2mpdn5",
       before: null
   }
}
```

There are a lot of properties for each `children` node here. But for our example we need only six, which are `permalink`, `subreddit`, `url`, `title`, `score`, and `num_comments`. These properties represent the permanent link on reddit for that post, the subreddit name to which the post belongs, title of the submitted post, overall score of the post, and the number of comments.

We simply iterate in the `children` array and the data for each post is present inside the `data` object of the element. We then create a list item for each post where we display the subreddit to which it belongs along with the title of post to its next. Clicking on the title will open the linked content which may be a reddit page or some external content. Next to the title we display the score of the post and number of comments. The comments text is a link which when clicked on opens the comments page on reddit in a new tab.

Once the whole DOM is created, we return it to the `initReddit` method where it is inserted in the div `#reddit`.

Now reload the `index.html` page in the browser and you will see the posts from reddit's front page:



reddit : top items

1 - [funny] To my fellow pale skinned individuals (score : 3630| comments : 310)

2 - [AdviceAnimals] At least I tried. (score : 3913| comments : 286)

3 - [pics] Andromeda's actual size if it were brighter (score : 4978| comments : 1538)

4 - [funny] Canadian Romeo & Juliet (score : 4171| comments : 421)

5 - [aww] MY cat's face before a bath (score : 2672| comments : 174)

6 - [funny] I pass this guy on my way to work.. so seductive. (score : 2923| comments : 70)

My mate franklin #kitten #franklin #ninja #mate(link)

tags : cute cat kitten

#kitten #relaxing #cute #cat(link)

tags : cute cat kitten

New York,USA

Temp: -1.87 degree celcius

Temp min: -4.00 degree celcius

Temp max: 0.00 degree celcius

Creating an image display widget

This is our last widget for this page and is very simple to implement. No AJAX is involved in it as well.

Before coding, we need to set up images that we will use in this widget. Create a folder named `images` inside the `Chapter11` folder. Inside the `images` folder, keep five images of your choice. For this example, I named them `1.jpg`, `2.jpg`, and so on. Create another folder named `thumb` inside the `images` folder. Now create small versions (preferably around 300 px wide) of these five images and place them in the `thumb` folder. With this we are good to proceed to the code.

We will create an array of images and their names, which we will display in a dropdown. Selecting an image will display its thumbnail. Clicking on the thumbnail will display its larger version in the jQuery UI dialog box.

Go to the `setupImageSelector` method of the dashboard object and write the following code that will set up the images array, display images names in dropdown, and bind the event handler to display images:

```
this.imageArr = [
  { id: 1,  name : 'Temple', path :  'images/1.jpg', thumb :
'images/thumb/1.jpg' },
  { id: 2, name : 'Colors', path :  'images/2.jpg', thumb :
'images/thumb/2.jpg' },
  { id: 3, name : 'Directions', path :  'images/3.jpg', thumb :
'images/thumb/3.jpg' },
  { id: 4, name : 'Flag', path :  'images/4.jpg', thumb :
'images/thumb/4.jpg' },
  { id: 5, name : 'A bit snow', path :  'images/5.jpg', thumb :
'images/thumb/5.jpg' }
];
var str = '<option value="0">select image</option>';
$.each(dashboard.imageArr, function(i, item)
{
  str+= '<option value="'+item.id+'">'+ item.name +'</option>'
});
$('#imageSelector').html(str);

$( "#imageSelector" ).on('change', function()
{
  dashboard.changeImage($(this));
});

$( "#thumbnail" ).on('click', function()
{
  var imgPath = $(this).data('large');
  $('#dialog').html('<img src="' + imgPath + '">').dialog({modal : true,
width: 'auto', top : 0});
});
```

We begin by defining the `imageArr` array. Each element in this array is an object containing image information. Each object has four properties: the `id` property that uniquely identifies an image, the `name` property of the image that will display the name of the image in a dropdown, the `path` property is the path of large images that can be a relative or absolute path, and lastly, the `thumb` property is the path to the thumbnail of the said image. Here we set the paths and names of images that we placed in the images folder.

Next, we loop in the images array and create options for dropdowns with value of each option as image ID. These options are then inserted into select box with `id` `imageSelector` of the last portlet.

After this, there are two event handlers. The first image handler is for the change event of dropdown. On the `change` event, it calls a `changeImage` method with selected options. To implement this method, add the following code to dashboard object:

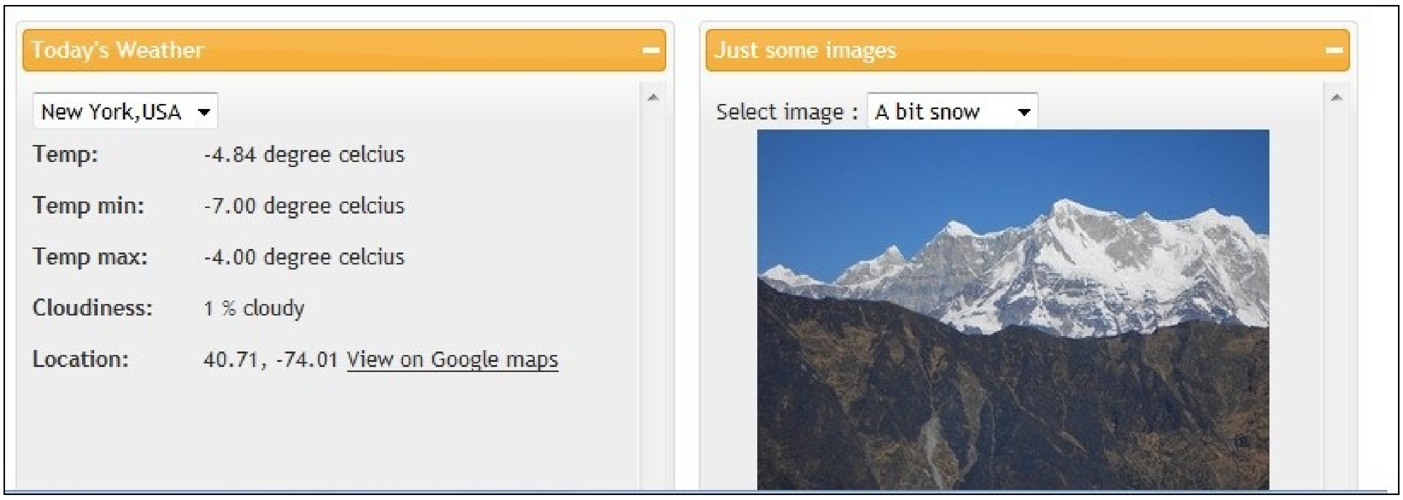
```
changeImage : function(selectedPic)
{
  if(parseInt(selectedPic.val(), 10) == 0)
  {
    $('#thumbnail').empty();
    return;
  }
  $.each(dashboard.imageArr, function(i, item)
  {
    if(parseInt(selectedPic.val(), 10) === item.id)
    {
      $('#thumbnail').data('large', item.path).html('<img src="' +
item.thumb +'">');
      return;
    }
  });
}
```

In the first line, we check the selected value. If it is `0`, we `empty` the div `#thumbnail` and exit from the method.

Next we iterate in `imageArr` and check the value of selected options against the `id` of each element in `imageArr`. Once a match is found, we add a data attribute `large` to the div `#thumbnail`, which contains paths for large images. Then we create an image element and set its `src` attribute to the thumbnail path `thumb`. Finally, this image is inserted inside div `#thumbnail`.

The second event handler inside `setupImageSelector` is to display the larger version of the image in a dialog. On clicking div `#thumbnail`, we first get the value of the data attribute `large`, which we defined in the preceding code. Then we create an image with `src` set to the image path and insert it inside the `#dialog` div. After this the jQuery UI's `dialog` method is called to display the dialog.

Check the `index.html` page in your browser now. Select an image from the dropdown and the thumbnail will be displayed. Clicking it will display the large version in a dialog:



Temp: -4.84 degree celcius

Temp min: -7.00 degree celcius

Temp max: -4.00 degree celcius

Cloudiness: 1 % cloudy

Location: 40.71, -74.01 View on Google maps

Improving the portlets

There is vast scope for improving all of the portlets. Here are some of the ideas to get you started:

- Use cookies to maintain the state of portlet positions so that positions of portlets remain intact even after the page loads
- Implement other sharing buttons like Pinterest, StumbleUpon, and so on
- Put a text box in the Flickr portlet and search for the API using user-entered tags
- In reddit portlet, display thumbnails of images in dialog
- Display the comments of a reddit post in a dialog
- In the weather portlet, allow users to enter cities
- Open Google Maps in-place in the weather portlet
- In the images portlet, instead of using a hardcoded images, pull the information from the database

Summary

In our last chapter of this book, we created some fun widgets using different APIs. I am sure you must have learned about integrating jQuery and jQuery UI components with different APIs.

As a bonus, there is an appendix following this chapter where you will learn about the powerful themeroller feature of jQuery UI using which you can create your own themes. We will also look at using different themeroller classes and using them in our projects.

I hope you had fun learning jQuery UI and creating different things in the 11 chapters of this book. Always keep learning new things; experimenting is the best way to do it.

Appendix A. Best Practices for Developing with jQuery UI

Congratulations for making it this far! We've had a great journey together in the 11 chapters of this book and I hope that you enjoyed reading and learning jQuery UI. In this final part of the book, we will discuss the best practices for using jQuery UI. Some techniques are applicable to jQuery UI in particular, while others apply to JavaScript and jQuery performance in general.

We will further divide this chapter into two sections. The first contains the general guidelines you should keep in mind while using jQuery UI, jQuery, and JavaScript in general. The second section contains the very useful jQuery UI themeroller. You will learn how we can create a new theme by customizing the look and feel of jQuery UI using themeroller.

General Guidelines

This section has some guidelines related to jQuery UI, jQuery, and JavaScript in general . Let’s go through them one by one.

Using a CDN

Instead of serving the jQuery UI library `.js` file and its theme's CSS file from your web server, it is better to use a **Content Delivery Network** (**CDN**). Using a CDN makes loading of web pages faster. This is because browsers cache `.js` files. Hence, if you have referenced a jQuery UI file from CDN, if it is found in the browser's cache from some earlier visit to a website, the browser will not load the file again. If you are using multiple libraries on a page, it can have a significant effect on reducing page-load time. Another advantage is that since these files are referenced from external domains, the browser will be able to make multiple requests to load the files.

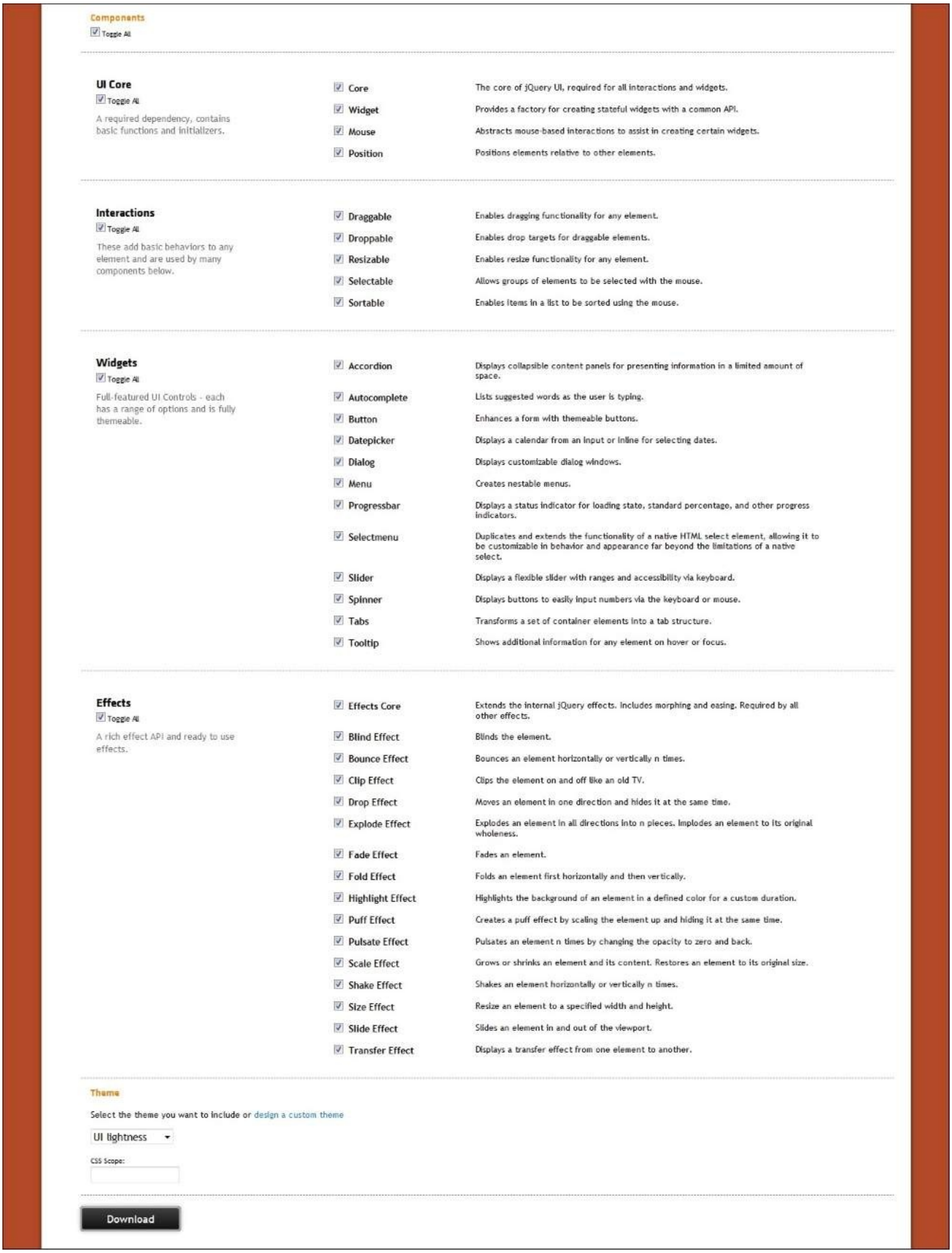
Here are the three popular and commonly used CDNs' for loading jQuery and jQuery UI files, along with their different versions:

- jQuery CDN: https://code.jquery.com/ui/
- CDN hosted by Google: https://developers.google.com/speed/libraries/devguide. Apart from jQuery and jQuery UI, Google CDN hosts several
- Microsoft CDN: http://www.asp.net/ajax/cdn

Use a customized build of jQuery UI

If you only want a tab widget or, say, a slider, it does not make sense to download the whole jQuery UI library just for this. The jQuery UI website has an option to create a customized build with only the required components.

Go to http://jqueryui.com/download/ in your browser, this will open a download builder page. Here you have two options, either download the complete build, or create a customized build:



Components

☑ Toggle All

UI Core

☑ Toggle All

A required dependency, contains basic functions and initializers.

☑ Core	The core of jQuery UI, required for all interactions and widgets.
☑ Widget	Provides a factory for creating stateful widgets with a common API.
☑ Mouse	Abstracts mouse-based interactions to assist in creating certain widgets.
☑ Position	Positions elements relative to other elements.

Interactions

☑ Toggle All

These add basic behaviors to any element and are used by many components below.

☑ Draggable	Enables dragging functionality for any element.
☑ Droppable	Enables drop targets for draggable elements.
☑ Resizable	Enables resize functionality for any element.
☑ Selectable	Allows groups of elements to be selected with the mouse.
☑ Sortable	Enables items in a list to be sorted using the mouse.

Widgets

☑ Toggle All

Full-featured UI Controls - each has a range of options and is fully themeable.

☑ Accordion	Displays collapsible content panels for presenting information in a limited amount of space.
☑ Autocomplete	Lists suggested words as the user is typing.
☑ Button	Enhances a form with themeable buttons.
☑ Datepicker	Displays a calendar from an input or inline for selecting dates.
☑ Dialog	Displays customizable dialog windows.
☑ Menu	Creates nestable menus.
☑ Progressbar	Displays a status indicator for loading state, standard percentage, and other progress indicators.
☑ Selectmenu	Duplicates and extends the functionality of a native HTML select element, allowing it to be customizable in behavior and appearance far beyond the limitations of a native select.
☑ Slider	Displays a flexible slider with ranges and accessibility via keyboard.
☑ Spinner	Displays buttons to easily input numbers via the keyboard or mouse.
☑ Tabs	Transforms a set of container elements into a tab structure.
☑ Tooltip	Shows additional information for any element on hover or focus.

Effects

☑ Toggle All

A rich effect API and ready to use effects.

☑ Effects Core	Extends the internal jQuery effects. Includes morphing and easing. Required by all other effects.
☑ Blind Effect	Blinds the element.
☑ Bounce Effect	Bounces an element horizontally or vertically n times.
☑ Clip Effect	Clips the element on and off like an old TV.
☑ Drop Effect	Moves an element in one direction and hides it at the same time.
☑ Explode Effect	Explodes an element in all directions into n pieces. Implodes an element to its original wholeness.
☑ Fade Effect	Fades an element.
☑ Fold Effect	Folds an element first horizontally and then vertically.
☑ Highlight Effect	Highlights the background of an element in a defined color for a custom duration.
☑ Puff Effect	Creates a puff effect by scaling the element up and hiding it at the same time.
☑ Pulsate Effect	Pulsates an element n times by changing the opacity to zero and back.
☑ Scale Effect	Grows or shrinks an element and its content. Restores an element to its original size.
☑ Shake Effect	Shakes an element horizontally or vertically n times.
☑ Size Effect	Resize an element to a specified width and height.
☑ Slide Effect	Slides an element in and out of the viewport.
☑ Transfer Effect	Displays a transfer effect from one element to another.

Theme

Select the theme you want to include or design a custom theme

UI lightness

CSS Scope:

For a customized build, first you have to select the version of jQuery UI you want to download. Next to it are listed all components, which are grouped in four sections. Each component has a checkbox next to it. Checking a component’s checkbox automatically selects other components that it depends on.

The first section is UI Core. You will probably want to retain all components as UI Core. It is the base on top of which all other components are built. Next are the interaction components: draggable, droppable, resizable, selectable, and sortable. All the components in this section require the Core, Widget, and Mouse components. You can verify this by unchecking all checkboxes first, and then checking any of the interaction components. The third section contains the widgets and the fourth and final section has the effects API.

Once you are done with the selections, proceed to the bottom of the page and you’ll see another option presented there to select a theme. At the time of writing, there are 2 dozen themes available. Select any one of these and click on **Download**. You will be prompted to download a customized build that will only contain your selected jQuery UI components.

Using the jQuery UI icons

jQuery UI also provides a large number of icons. Displaying an icon is also very easy. To display an icon you just have to give two CSS class names to an HTML element. For a particular icon, “ui-icon” is followed by the class name of that icon. The ui-icon is the base class which must be included for each icon to be displayed.

Go to the page http://api.jqueryui.com/theming/icons/ to see all available icons.

Be specific about elements as much as possible

Element IDs are the fastest to search for in DOM compared to class names and element names, since they are unique. You will not notice any difference in speed while using any of the methods in small pages. But say you have a large page with hundreds of table rows that are dynamically created. you will start seeing lags if your selectors are not efficient.

For example let’s have a look at the following HTML code:

```
<div class="outerDiv">
  <div class="innerDiv">
    <p>A paragraph</p>
    <ul id="myList">
      <li>first item</li>
      <li>second item</li>
      <li>third item</li>
    </ul>
  </div>
</div>
```

Now suppose you want to hide the `ul` element. We can use any of the two ways as written below:

```
//Method 1
$('#myList').hide();
//Method 2
$('.outerDiv').find('ul#myList').hide();
```

For a page with a large number of elements, `Method 1` in the preceding code will be fastest as opposed to `Method 2`.

The point here is to be as precise as you can. If you have a choice between using multiple parent selectors and a find selector method, always go for find.

Chain, chain, and chain

Chaining is a great feature in jQuery, and we have used it in past chapters as well. Almost all jQuery methods return the element itself, when a method is called. This can be used to call multiple methods on an element. The advantages here are performance, avoiding repetition, readability, and significant reduction in the number of lines.

Suppose you have an HTML element and you want to remove an existing class from it, add a new class, and then toggle it. First let us do it without chaining, as shown in the following code snippet:

```
$('#element').removeClass('oldClass');
$('#element').addClass('newClass');
$('#element').slideToggle('slow');
```

Those are three lines and a lot of repetition. Now let’s use chaining and do the same again:

```
$('#reset').removeClass('oldClass')
  .addClass('newClass')
  .slideToggle('slow');
```

Now this is a single statement. You will encounter these types of situations regularly in your applications, where chaining can save you a lot of typing as well as improve performance.

Cache selectors

Whenever a selector is used, DOM is searched for that element and then the element is fetched. As a rule of thumb, you should avoid touching the DOM as much as possible. Caching selectors can help in this situation; once a selector is retrieved from DOM, it can be stored in a JavaScript variable and any jQuery or jQuery UI methods can be used on it. Here is an example:

```
$('#dialog').dialog();

$('#dialog').on('dialogopen', function(event,ui)
{
  //do something here
});
```

Instead of searching for an element with ID dialog again, you can cache it and use it again like this:

```
var dialog = $('#dialog');

dialog.dialog();

dialog.on('dialogopen', function(event,ui)
{
  //do something here
});
```

The preceding code retrieves the element from DOM only once and then acts on it as many times as needed.

Cache your loop variables

Similarly to that mentioned previously, when you are running `for` loops, you can cache those as well:

```
Var myArray;//an array of 1000 items
for(var i=0; i<myArray.length; i++)
{
  //do something here
}
```

What happens is that the length of `myArray` is calculated in each iteration, which is obviously not good. The solution is simple, calculate length beforehand and use that value:

```
Var myArray;
for(var i=0, len = myArray.length; i < len; i++)
{
  //do something here
}
```

While the loop is initializing, we store the array length in a variable len and this variable is used for all iterations.

Note

Small arrays may not have a visible effect on this but this is a recommended practice for larger arrays.

DOM manipulation is costly

Each time an element is removed from DOM, added to it, or any change is made to an existing element, the whole DOM is affected. This is called reflow. While using jQuery we do not think of it often but each `addClass`, `css`, `html`, `text`, and `append` method causes the DOM to reflow.

A simple example is iterating in an array and creating an unordered list:

```
//assuming you have a ul with id myList in DOM
var fruits = ['Apple', 'Orange', 'Banana', 'Guava', 'Mango'];
for(var i =0; i< fruits.length; i++)
{
    $('#myList').append('<li>' + fruits[i] + '</li>');
}
```

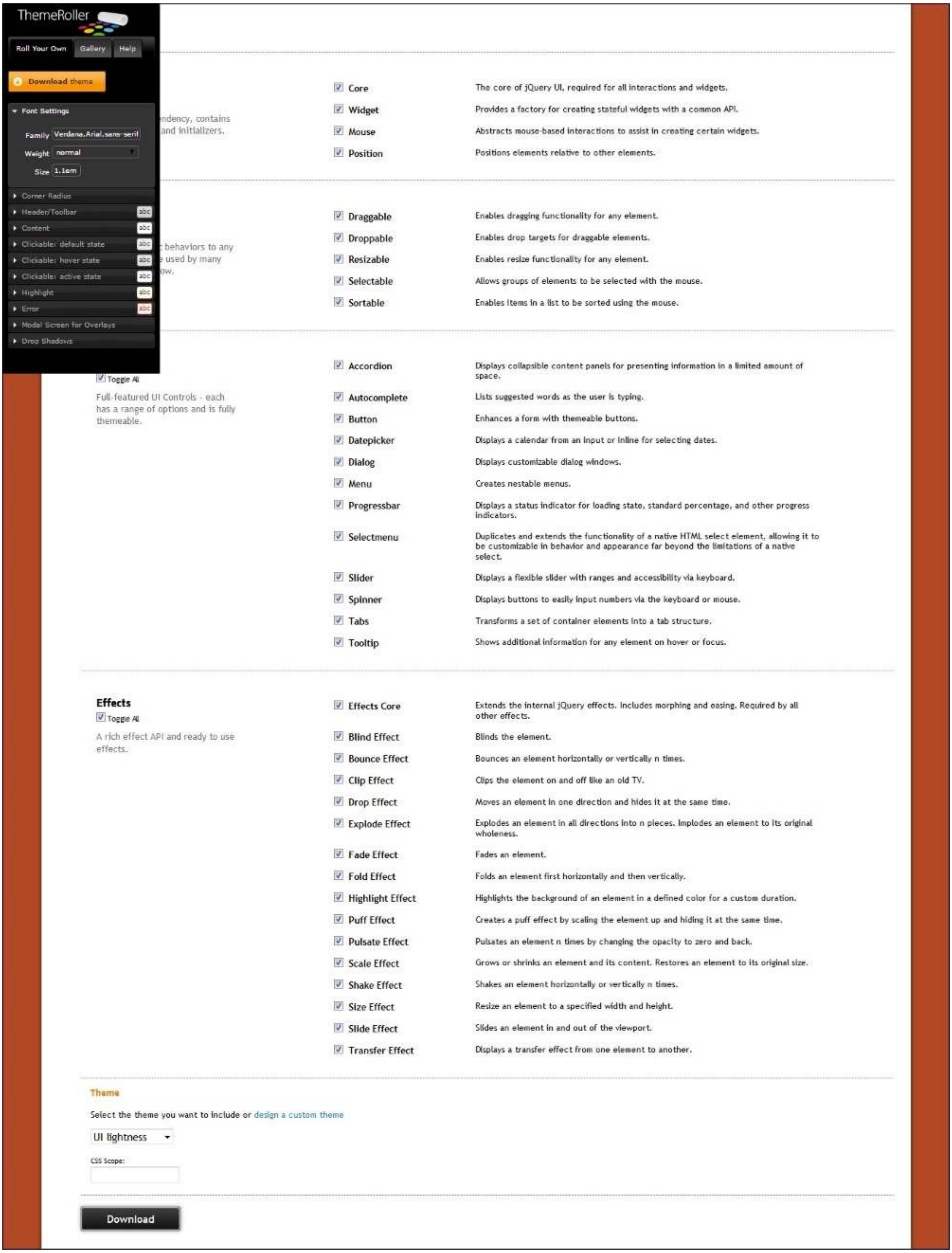
The preceding code looks completely harmless but it’s reflowing the DOM five times. With some simple changes, we can reduce this number to one. Here’s how:

```
//assuming you have a ul with id myList in DOM
var fruits = ['Apple', 'Orange', 'Banana', 'Guava', 'Mango'];
var listElement = $('#myList');
var strDOM = '';

for(var i =0; i< fruits.length; i++)
{
  strDOM+= '<li>' + fruits[i] + '</li>';
}

listElement.html(strDOM);
```

We created the DOM and stored it in a local variable. Once the loop is over we pushed it back to the DOM.

Using jQuery UI Themeroller to customize a theme

jQuery UI themeroller is a great utility to design your own jQuery UI theme. It allows you to customize the look and feel of the theme completely. The themeroller page can be found at http://jqueryui.com/themeroller/. On opening this page, you will find a box on the left-hand side with three tabs. The first tab is called **Roll Your Own**, the second tab is **Gallery**, which has 24 prebuilt themes available, and the third tab is **Help**. We are going to look at the first tab and its different options to customize the theme. Here is how the first tab looks:



ThemeRoller

Roll Your Own | Gallery | Help

Download theme

Font Settings

Family: Verdana,Arial,sans-serif

Weight: normal

Size: 1.1em

Corner Radius

Header/Toolbar

Content

Clickable: default state

Clickable: hover state

Clickable: active state

Highlight

Error

Modal Screen for Overlays

Drop Shadows

ndency, contains
and initializers.

☑ Core	The core of jQuery UI, required for all interactions and widgets.
☑ Widget	Provides a factory for creating stateful widgets with a common API.
☑ Mouse	Abstracts mouse-based interactions to assist in creating certain widgets.
☑ Position	Positions elements relative to other elements.

behaviors to any
e used by many
ow.

☑ Draggable	Enables dragging functionality for any element.
☑ Droppable	Enables drop targets for draggable elements.
☑ Resizable	Enables resize functionality for any element.
☑ Selectable	Allows groups of elements to be selected with the mouse.
☑ Sortable	Enables items in a list to be sorted using the mouse.

☑ Toggle All

Full-featured UI Controls - each has a range of options and is fully themeable.

☑ Accordion	Displays collapsible content panels for presenting information in a limited amount of space.
☑ Autocomplete	Lists suggested words as the user is typing.
☑ Button	Enhances a form with themeable buttons.
☑ Datepicker	Displays a calendar from an input or inline for selecting dates.
☑ Dialog	Displays customizable dialog windows.
☑ Menu	Creates nestable menus.
☑ Progressbar	Displays a status indicator for loading state, standard percentage, and other progress indicators.
☑ Selectmenu	Duplicates and extends the functionality of a native HTML select element, allowing it to be customizable in behavior and appearance far beyond the limitations of a native select.
☑ Slider	Displays a flexible slider with ranges and accessibility via keyboard.
☑ Spinner	Displays buttons to easily input numbers via the keyboard or mouse.
☑ Tabs	Transforms a set of container elements into a tab structure.
☑ Tooltip	Shows additional information for any element on hover or focus.

Effects

☑ Toggle All

A rich effect API and ready to use effects.

☑ Effects Core	Extends the internal jQuery effects. Includes morphing and easing. Required by all other effects.
☑ Blind Effect	Blinds the element.
☑ Bounce Effect	Bounces an element horizontally or vertically n times.
☑ Clip Effect	Clips the element on and off like an old TV.
☑ Drop Effect	Moves an element in one direction and hides it at the same time.
☑ Explode Effect	Explodes an element in all directions into n pieces. Implodes an element to its original wholeness.
☑ Fade Effect	Fades an element.
☑ Fold Effect	Folds an element first horizontally and then vertically.
☑ Highlight Effect	Highlights the background of an element in a defined color for a custom duration.
☑ Puff Effect	Creates a puff effect by scaling the element up and hiding it at the same time.
☑ Pulsate Effect	Pulsates an element n times by changing the opacity to zero and back.
☑ Scale Effect	Grows or shrinks an element and its content. Restores an element to its original size.
☑ Shake Effect	Shakes an element horizontally or vertically n times.
☑ Size Effect	Resize an element to a specified width and height.
☑ Slide Effect	Slides an element in and out of the viewport.
☑ Transfer Effect	Displays a transfer effect from one element to another.

Theme

Select the theme you want to include or design a custom theme

UI lightness

CSS Scope:

Download

Changing any value in this left-hand side box instantly changes the look of the elements on the page, so you can see how the theme is going to look.

Let's look at the different options available to customize the theme:

- **Global styles**: The following are the global styles that can be applied:
    - **Font settings**: These settings specify the global font name and the font size. If you are planning to use a custom font (for example, the Google font), just type the name here and load the font separately.
    - **Corner radius**: This is the border radius width for all elements. Its default value is 4 px.
- **Generic**: The **Header/Toolbar**, **Content**, various **Clickable states**, **Highlight**, and **Error** have common fields that are as follows:
    - **Background color and texture**: You can specify a hex code for a color or select a color from the palette. The texture value is provided as a percentage.
    - **Border, text, and icon colors**: As the name suggests, you have to provide color codes for the border color, text color, and the color of the icon.
- **Others**: We have a few more options to customize our theme:
    - **Modal screens**: For modal overlays, you can specify the background color and texture. For the overlay, you can also specify its opacity.
    - **Shadows**: For drop shadows on overlays, you can specify the background color and texture. Besides this, there are also options to specify the shadow thickness and offset.

Once you are done with setting all the options and are satisfied with the live preview of the theme, just click the **Download Theme** button at the top and your customized theme will be ready to download to your machine.

Index

A

- accept option / Handling events for puzzle pieces
- accordion component, properties
  - collapsible / Displaying hotels in accordion
  - active / Displaying hotels in accordion
  - heightStyle / Displaying hotels in accordion
- ajaxAlbum.php file, parameters
  - action / Editing a picture name
  - albumId / Editing a picture name
  - pictureId / Editing a picture name
  - newImageName / Editing a picture name
- album names, photo album manager
  - filling / Filling album names
  - albums, displaying / Displaying the albums
  - sortable pictures, creating / Making the pictures sortable
  - dialogs, initializing for edit / Initializing dialogs for edit, delete, and zoom
  - dialogs, initializing for delete / Initializing dialogs for edit, delete, and zoom
  - dialogs, initializing for zoom / Initializing dialogs for edit, delete, and zoom
  - click events, handling for delete icon / Handling click events for edit, delete icons, and zooming pictures
  - click events, handling for edit / Handling click events for edit, delete icons, and zooming pictures
  - click events, handling for image zoom / Handling click events for edit, delete icons, and zooming pictures
- albums object, properties
  - jsonAlbums / Getting code structure ready
  - currentAlbum / Getting code structure ready
  - currentPictureId / Getting code structure ready
  - initialize / Getting code structure ready
  - fillAlbumNames / Getting code structure ready
  - addEventHandlers / Getting code structure ready
  - displayAlbum / Getting code structure ready
  - editImage / Getting code structure ready
  - deleteImage / Getting code structure ready
  - saveNewSequence / Getting code structure ready
- API key, Google Maps
  - obtaining / Getting a Google Maps API key
  - URL / Getting a Google Maps API key

C

- CAPTCHA
  - folder structure, creating / Creating the folder structure
- CDN
  - about / Using jQuery/jQuery UI libraries with a CDN, Using a CDN
  - using / Using a CDN
- CDNJS
  - URL / Using jQuery/jQuery UI libraries with a CDN
- chaining
  - about / Chain, chain, and chain
- Chrome DevTools
  - URL / The CSS background-position property
- code structure, dashboard
  - defining / Getting the code structure ready
- color picker
  - folder structure, setting up / Setting up the folder structure
  - markup, writing for page / Writing markup for the page
  - content, styling / Styling the content
  - implementing / Implementing the color picker
  - object structure, defining / Defining the object structure
  - init method, initializing / The init method
  - colors, modifying / Changing colors and updating the spinner
  - spinner, updating / Changing colors and updating the spinner
  - getHexColor method, defining / The getHexColor method
  - convertToHex method, defining / Converting to hex
  - improving / Improving the Colorpicker
- colorPicker object
  - init method / Defining the object structure
  - setColor method / Defining the object structure
  - getHexColor method / Defining the object structure
  - convertToHex method / Defining the object structure
- createSlices function / Creating slices of the image
- CSS background-position property
  - about / The CSS background-position property
- CSS rule declarations, properties
  - body / Styling the content
  - .container / Styling the content
  - #timeline / Styling the content
  - .year / Styling the content
  - .year div / Styling the content
  - #window / Styling the content
  - .close / Styling the content
  - .overlay / Styling the content

    - #leftOverlay / Styling the content
    - #rightOverlay / Styling the content
    - #yearEvents ul li / Styling the content
    - #slider / Styling the content
    - #sliderVal / Styling the content
    - .clear / Styling the content
- customized build, jQuery UI
    - using / Use a customized build of jQuery UI
    - URL / Use a customized build of jQuery UI

D

- dashboard
  - stage, setting up / Setting up the stage
  - page, designing / Designing the page
  - code structure, defining / Getting the code structure ready
  - portlets, initializing / Initializing the portlets
  - sharing buttons, implementing / Implementing sharing buttons
  - Flickr photos, displaying / Displaying Flickr photos
  - weather widget, creating / Creating a weather widget
  - posts, displaying from reddit front page / Displaying posts from the reddit front page
  - image display widget, creating / Creating an image display widget
  - portlets, improving / Improving the portlets
- DOM
  - about / Displaying data on the page
- drag and drop CAPTCHA
  - implementing / Implementing the drag and drop CAPTCHA
  - random color, setting / Setting a random color
  - displaying / Displaying the CAPTCHA
  - making functional / Making the drag and drop functional
  - server, validating / Validating on the server
  - color CAPTCHA, improving / Improving the color CAPTCHA
- draggable method, options
  - revert / Draggable country names
  - revertDuration / Draggable country names
  - cursor / Draggable country names
- drop option / Handling events for puzzle pieces
- droppable capital names, quiz application
  - creating / Droppable capital names and scoring
  - draggable element, accepting / Accepting a draggable element
  - drop event, using / The drop event

E

- event timeline
  - folder structure, creating / Creating the folder structure
  - HTML markup, writing / Designing the page
  - content, styling / Styling the content
  - code structure / Getting the code structure ready
  - markup, creating from data / Creating the timeline markup from data
  - functionality, implementing / Implementing the timeline functionality
  - improving / Improving the timeline

F

- Firebug
  - URL / The CSS background-position property
- Flickr photos, dashboard
  - displaying / Displaying Flickr photos
- folder structure
  - setting up / Setting up the folder structure for the JavaScript and CSS files
- functionality, event timeline
  - slider, implementing / Making the slider work
  - slider, implementing with if block / The if block
  - slider, implementing with else block / The else block
  - year window, dragging / Dragging the year window
  - event details, displaying on year window click / Displaying event details when a year window is clicked on
  - event details window, closing / Closing the event details window
- functional puzzle, jigsaw puzzle game
  - creating / Making the puzzle functional
  - image slices, creating / Creating slices of the image
  - CSS background-position property / The CSS background-position property
  - events, handling for puzzle pieces / Handling events for puzzle pieces
  - puzzle completion, checking for / Checking for puzzle completion
  - resetting / Resetting the puzzle

G

- Google CDN
    - URL / Using jQuery/jQuery UI libraries with a CDN, Using a CDN
- guidelines, jQuery UI
    - about / General Guidelines
    - CDN, using / Using a CDN
    - icons, using / Using the jQuery UI icons
    - elements, specifying / Be specific about elements as much as possible
    - chaining / Chain, chain, and chain
    - selectors, caching / Cache selectors
    - variables, caching / Cache your loop variables
    - DOM manipulation / DOM manipulation is costly

H

- home page, tabbed news reader
  - designing / Designing the page
  - markup, writing for tabs / Writing markup for tabs
  - content, styling / Styling the content
- home page, website tour
  - designing / Designing the home page
  - markup, writing / Writing markup for the page
  - elements, styling / Styling elements
- hoverClass option / Handling events for puzzle pieces

I

- icons
  - URL / Designing the page
- icons, jQuery UI
  - using / Using the jQuery UI icons
  - URL / Using the jQuery UI icons
- image display widget, dashboard
  - creating / Creating an image display widget

J

- jigsaw puzzle game
  - building / Getting ready
  - layout, creating / Creating the layout
  - functional puzzle, creating / Making the puzzle functional
  - Start button, implementing / Starting the game
  - puzzle, improving / Improving the puzzle
- jQuery
  - URL, for downloading / Downloading the required files
  - downloading / Downloading the required files
- jQuery/jQuery UI libraries
  - using, with CDN / Using jQuery/jQuery UI libraries with a CDN
- jQuery CDN
  - URL / Using a CDN
- jQuery UI
  - setting up / Setting up jQuery UI
  - URL, for downloading / Downloading the required files
  - downloading / Downloading the required files
  - folder structure, setting up / Setting up the folder structure for the JavaScript and CSS files
  - guidelines / General Guidelines
  - customized build, using / Use a customized build of jQuery UI
- jQuery UI sortable example
  - URL / Designing the page

L

- layout, jigsaw puzzle game
  - creating / Creating the layout
  - markup, creating / Creating the markup for the puzzle
  - elements, styling / Styling elements

M

- mapOptions object, properties
  - center / Displaying the map
  - zoom / Displaying the map
  - disableDefaultUI / Displaying the map
  - mapTypeId / Displaying the map
  - scrollwheel / Displaying the map
- maps mashup
  - folder structure, creating / Creating the folder structure
  - Google Maps API key, obtaining / Getting a Google Maps API key
  - page, designing / Designing the page
  - content, styling / Styling the content
  - code structure / Getting the code structure ready
  - minimum and maximum prices, setting / Setting minimum and maximum prices
  - hotels, displaying in accordion / Displaying hotels in accordion
  - spinner, setting up / Setting up the spinner
  - map, displaying / Displaying the map
  - markers, setting in map / Setting markers and infowindows in the map
  - infowindows, setting in map / Setting markers and infowindows in the map
  - slider, implementing / Implementing the slider
  - improving / Improving the functionality
- Microsoft CDN
  - URL / Using jQuery/jQuery UI libraries with a CDN, Using a CDN
- myMap object, properties
  - map / Getting the code structure ready
  - markers / Getting the code structure ready
  - infowindow / Getting the code structure ready
  - minPrice / Getting the code structure ready
  - maxPrice / Getting the code structure ready
  - hotelsList / Getting the code structure ready
  - initialize / Getting the code structure ready
  - setMinMaxPrices / Getting the code structure ready
  - displayHotels / Getting the code structure ready
  - setSpinner / Getting the code structure ready
  - createMap / Getting the code structure ready
  - setMarkersAndInfoWindow / Getting the code structure ready
  - setSlider / Getting the code structure ready

N

- number CAPTCHA
  - creating / Creating the number CAPTCHA
  - five-digit number, generating / Generating the five-digit number
  - displaying, on page / Displaying CAPTCHA on the page
  - sortable functionality, adding / Adding the sortable functionality
  - number, validating on server / Validating the number on the server

O

- object literal
  - about / Initializing the tour
- objTimeline object, properties
  - itemsToDisplay / Getting the code structure ready
  - minYear / Getting the code structure ready
  - maxYear / Getting the code structure ready
  - currentYear / Getting the code structure ready
  - timelineWindowStartYear / Getting the code structure ready
  - windowLeft / Getting the code structure ready
  - isWindowOpen / Getting the code structure ready
  - timelineData / Getting the code structure ready
  - init / Getting the code structure ready
  - createMarkup / Getting the code structure ready
  - createTimeline / Getting the code structure ready
  - closeWindow / Getting the code structure ready

P

- page, dashboard
  - designing / Designing the page
  - content, styling / Styling the content
- page, photo album manager
  - designing / Designing the page
  - placeholders, creating for albums / Creating placeholders for albums and pictures
  - placeholders, creating for pictures / Creating placeholders for albums and pictures
  - markup, writing for dialog boxes / Writing markup for dialog boxes
  - content, styling / Styling the content
- photo album manager
  - folder structure, creating / Creating the folder structure
  - page, designing / Designing the page
  - JSON file, creating for albums / Creating the JSON file for albums
  - code structure / Getting code structure ready
  - initialize method, implementing / Implementing the initialize method
  - album names, filling / Filling album names
  - improving / Improving album manager
- pictures, photo album manager
  - editing / Editing, deleting, and rearranging pictures
  - deleting / Editing, deleting, and rearranging pictures, Deleting a picture
  - rearranging / Editing, deleting, and rearranging pictures, Rearranging pictures of an album
  - picture name, editing / Editing a picture name
  - ajaxAlbum.php file / The ajaxAlbum.php file
- pictures array, properties
  - id / Creating the JSON file for albums
  - sequence / Creating the JSON file for albums
  - imageTitle / Creating the JSON file for albums
  - imageThumb / Creating the JSON file for albums
  - imageLarge / Creating the JSON file for albums
- portlets, dashboard
  - initializing / Initializing the portlets
- posts, dashboard
  - displaying, from reddit front page / Displaying posts from the reddit front page
- posts, tabbed news reader
  - displaying, of subreddit / Displaying posts of a subreddit
  - tab structure, creating / Creating the tab structure
  - DOM, building / Building the DOM for posts
  - comments, obtaining / Getting comments for a post

Q

- quiz application
  - layout, creating / Creating the layout
  - index.html file, editing / Markup for the quiz page
  - elements, styling / Styling elements
  - making functional / Making the quiz functional
  - data, displaying / Displaying data on the page
  - draggable country names, creating / Draggable country names
  - droppable capital names, creating / Droppable capital names and scoring
  - scores, adding / Droppable capital names and scoring
  - resetting / Resetting the quiz
  - enhancing / Improving the quiz

R

- range slider, properties
  - min / Implementing the slider
  - max / Implementing the slider
  - range / Implementing the slider
  - values / Implementing the slider
  - step / Implementing the slider
  - slide / Implementing the slider
  - stop / Implementing the slider
- reflow
  - about / DOM manipulation is costly

S

- searchable widget
  - folder structure / The folder structure
  - creating / Creating a widget to search data in a table
  - markup, writing for table / Writing markup for the table
  - content, styling / Styling the content
  - implementing / Implementing the widget
  - widget structure, defining / Defining the widget structure
  - default options, setting up / Setting up default options
  - initializing / Initializing the widget and attaching event handlers
  - event handlers, attaching / Initializing the widget and attaching event handlers
  - table, filtering / Filtering the table
  - option value, setting / Making changes when the option value is changed
  - destroying / Destroying the widget
  - calling / Calling the widget from the page
  - improving / Improving the searchable widget
- sharing buttons implementation, dashboard
  - about / Implementing sharing buttons
  - Facebook / Implementing sharing buttons
  - Twitter / Implementing sharing buttons
  - Reddit / Implementing sharing buttons
  - Google Plus / Implementing sharing buttons
- shuffle function
  - URL / Displaying data on the page
- slider CAPTCHA
  - creating / Creating the slider CAPTCHA
  - minimum value, generating / Generating minimum and maximum values for the slider
  - maximum value, generating / Generating minimum and maximum values for the slider
  - making functional / Making the slider functional
  - slider values, validating / Validating the slider values
  - improving / Improving the slider CAPTCHA
- slide widget
  - folder structure / The folder structure
  - creating / Creating a widget to display a slideshow
  - markup, writing / Writing markup for slides
  - content, styling / Styling the content
  - implementing / Implementing the widget
  - widget structure, defining / Defining the widget structure
  - default options, setting up / Setting up default options
  - initializing / Initializing the widget and displaying the first slide
  - first slide, displaying / Initializing the widget and displaying the first slide

  - slides, displaying one by one / Displaying slides one by one
  - option value, setting / Making changes when the option value is changed
  - destroying / Destroying the widget
  - calling / Calling the widget from the page
  - improving / Improving the banner widget
- slide widget, options
  - duration / Setting up default options
  - effect / Setting up default options
  - easingDuration / Setting up default options
  - beforeShow / Setting up default options
- sortable component
  - connectWith option / Initializing the portlets
  - handle option / Initializing the portlets
  - cancel option / Initializing the portlets
  - placeholder option / Initializing the portlets
- sortable components, properties
  - handle / Making the pictures sortable
  - placeholder / Making the pictures sortable
  - cursor / Making the pictures sortable
- stage, dashboard
  - setting up / Setting up the stage

T

- tabbed news reader
  - folder structure, creating / Creating the folder structure
  - home page, designing / Designing the page
  - code structure / Getting the code structure ready
  - event handlers, adding in init method / Adding event handlers in the init method
  - posts, displaying of subreddit / Displaying posts of a subreddit
  - improving / Improving MyjqReddit
- themeroller
  - about / Using jQuery UI Themeroller to customize a theme
  - used, for customizing theme / Using jQuery UI Themeroller to customize a theme
  - URL / Using jQuery UI Themeroller to customize a theme
  - options / Using jQuery UI Themeroller to customize a theme

W

- $.widget() function, properties
  - options / Defining the widget structure
  - _create / Defining the widget structure
  - _destroy / Defining the widget structure
  - _setOption / Defining the widget structure
- weather widget, dashboard
  - creating / Creating a weather widget
  - URL / Creating a weather widget
- website tour
  - folder structure, creating / Getting ready
  - home page, designing / Designing the home page
  - functionality, implementing / Making the tour functional
  - improving / Improving the tour
- website tour, functionality
  - implementing / Making the tour functional
  - accordion, initializing / Initializing accordion and tooltips
  - tooltips, initializing / Initializing accordion and tooltips
  - dialog box, initializing / Defining the dialog
  - tour steps, defining / Defining the tour steps
  - initializing / Initializing the tour
  - tour step, displaying / Displaying a tour step
  - Next button, defining / Making the Previous and Next buttons functional
  - Previous button, defining / Making the Previous and Next buttons functional
  - ending / Ending the tour