INSTANT

Short | Fast | Focused

HTML5 Local Storage How-to

Over 15 recipes to take advantage of the HTML5 Local Storage standard

Alex Libby

[PACKT] PUBLISHING

Instant HTML5 Local Storage How-to

Over 15 recipes to take advantage of the HTML5 Local Storage standard

Alex Libby

[PACKT]
PUBLISHING

BIRMINGHAM - MUMBAI

Instant HTML5 Local Storage How-to

First published: March 2013

Production Reference: 1190313

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-84969-931-0

www.packtpub.com

Credits

Author
Alex Libby

Reviewer
Laurentiu Nicolae

Acquisition Editor
Kartikey Pandey

Commissioning Editor
Maria D'souza

Technical Editor
Prasad Dalvi

Project Coordinator
Esha Thakker

Proofreader
Ting Baker

Production Coordinator
Prachali Bhiwandkar

Cover Work
Prachali Bhiwandkar

Cover Image
Valentina D'Silva

About the Author

Alex Libby works in IT support. He has been involved in supporting end users for the last 15 years in a variety of different environments, and currently works as a Technical Analyst, supporting a medium-sized SharePoint estate for an international parts distributor based in the U.K. Although Alex gets to play with different technologies in his day job, his first true love has always been with the open source movement, and in particular experimenting with CSS3 and HTML5. To date, Alex has written several books for Packt Publishing, including one on HTML5 Video and another on JQuery Tools. *HTML5 Local Storage How-To*, is his fifth book.

I'd like to thank my family and friends for their help and encouragement, Maria for her help and guidance in writing the book, and Laurentiu Nicolae for providing lots of constructive comments with the reviewing. Without them, I am sure I wouldn't have been able to produce this book!

About the Reviewer

Laurentiu Nicolae is a Senior Web Developer with over six years of experience. He has been working as a freelancer for different clients from all over the world. He is a native writer in web languages such as PHP, HTML, CSS, and JavaScript. He lives in Bucharest, Romania, with his wife, Georgiana, and their son, Matei.

www.PacktPub.com

Support files, eBooks, discount offers and more

You might want to visit `www.PacktPub.com` for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at `www.PacktPub.com` and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at `service@packtpub.com` for more details.

At `www.PacktPub.com`, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.

`http://PacktLib.PacktPub.com`

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

Why Subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print and bookmark content
- On demand and accessible via web browser

Free Access for Packt account holders

If you have an account with Packt at `www.PacktPub.com`, you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

Table of Contents

Preface

A question: how many times have you visited a site, where it says "Hello,..." followed by your name? A few hundred...? A thousand...? Lost count...?

I have no doubt that you've probably visited sites, such as Amazon, where this innocuous piece of text is very likely to be displayed using cookies. We've grown accustomed to seeing text such as this everywhere, but it was Netscape's pioneering work back in 1994 that made this possible.

But, I hear you ask, what do cookies have to do with Local Storage? It's simple, cookies are the forerunners for what we now know as Local Storage. In their heyday, cookies worked well, but they have now become old technology. What if I was to say that you could store more than simple text in a browser? How about images? You're probably thinking this isn't possible, but we will see how you can do this, and more, throughout this book.

HTML5 Local Storage was designed to replace cookies, largely as a way to alleviate privacy and security concerns, but also to allow you to do much more while also reducing requests to the server. As a bonus, it also works in most recent browsers, such as IE 8 or recent versions of Firefox. If you still need to support older ones, this isn't a problem, as you will see later in this book!

Interested in learning more? Let's get started...

What this book covers

Throughout this book, we're going to look at a variety of exercises that are designed to help you get accustomed to working with the basics of the new HTML5 Local Storage standard. You're probably wondering what we're going to cover, right? No problem, let me reveal all.

Basic use of Local Storage (*Must know*) discusses the basics around how you can use HTML5 Local Storage, the security aspects that you need to be aware of, and some of the limitations of what it can and can't do.

Viewing Local Storage content in a browser (*Must know*) explains how you can view Local Storage content in your browser. It is easy, and you'll see how, using tools such as FireStorage or Web Developer Toolbar in your browser.

Basic demo for Session Storage (*Must know*) discusses Session Storage, which is perfect for instances when you only need to store values temporarily.

Comparing Local Storage and Session Storage (*Should know*) provides an example of how you can use Local Storage and Session Storage together.

Using web storage instead of cookies (*Must know*) illustrates how you can approximate the effect of Session Storage when using old browsers.

Basic detection for storage support (*Must know*) explains how to check if older browsers will work.

Improving detection using Modernizr (*Should know*) takes you through how you can improve the checking, using something like Modernizr.

Providing fallback support (*Should know*) explains how to use fallback support when your browser is not supporting Local Storage.

Is the user online or offline? (*Become an expert*) discusses how you can check whether you are connected to the Internet or not when using Local Storage.

Using a manifest for caching (*Become an expert*) illustrates how you can use a manifest file to implement caching and improve response times.

Providing support for the mobile platform (*Should know*) explains how you can provide support for mobile phones or tablets.

Building a simple plugin using jQuery (*Should know*) shows you how you can use the power of jQuery to build a basic plugin that you can re-use in your projects.

Adding objects, arrays, and TTL support to storage (*Become an expert*) explains how you can include support for objects, arrays, and a "time-to-live" facility, which is not available using the standard functionality.

Storing images within Local Storage (*Become an expert*) explains how you can store images if using jQuery. This will also help in improving response times.

Adjusting the Local Storage space for browsers (*Must know*) discusses how to cope with the maximum allowed space 5 MB.

Building a stickies option for using in a browser (*Become an expert*) provides the demos that will show you how to use Local Storage by building a simple stickies applet that you can use in your browser, when viewing websites.

Building a simple to-do list (*Become an expert*) illustrates how to build a simple to-do list that you can use as the basis for your own projects.

Using Local Storage in a form (*Become an expert*) explains how to use Local Storage to fill up forms. In this recipe, we move onto a two-part demo, looking at where Local Storage can really make a difference.

Using Local Storage in a CMS (*Become an expert*) explains how to adapt Local Storage to be used in a CMS, such as WordPress.

Using Local Storage to hide sign-up forms (*Become an expert*) explains how you can use Local Storage to prevent you from getting irritated in the situations when you use a site that insists on displaying a prompt to sign up, even though you are already a registered user.

What you need for this book

There will be instances where you may need to install software for a particular recipe. We will go through the specifics of each piece of software, ahead of any task. In the meantime, you will need the following:

- An Internet connection (for downloading various pieces of software for each chapter).
- A working installation of WordPress, for the form demo towards the end of the book.
- A modern browser, which must be one capable of running CSS3. Ideal examples would be the latest versions of Firefox, Safari, Chrome, or IE. We will look at backward compatibility in older browsers, but the effect will not be the same!
- A text editor—there are hundreds available for free or low cost. Alternatively, you can use something like Notepad. My personal preference is Textpad, a shareware application, which is available at `http://www.textpad.com`.

Who this book is for

This book is great for those who are new to the HTML5 Local Storage standard. You are likely to have some experience of using jQuery already, and run a site that could benefit from using the new functionality. I'll take you through how you can implement the new standard, and provide tips and some examples that you can either drop in or adapt for future projects.

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 are shown as follows: "Then, add the following code and save it as a test `localdemo.html`"

A block of code is set as follows:

```
<!DOCTYPE html>
<html>
  <head>
    <script src="http://code.jquery.com/jquery-1.8.1.min.js"></script>
    <script type="text/javascript">
    </script>
  </head>
  <body>
  </body>
</html>
```

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

```
<script type="text/javascript">
  function storeItem() {
    var item = $('#item').val();

    var items = localStorage.getItem('myItems');
      if (items != null) {
        items = JSON.parse(items);
      } else {
      items = new Array();
      }

    items.push(item);

    localStorage.setItem('myItems',
    JSON.stringify(items));

    refresh();
  }
</script>
```

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: "Click on the **Add to Firefox** button."

Warnings or important notes appear in a box like this.

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 book that you need and would like to see us publish, please send us a note in the **SUGGEST A TITLE** form on `www.packtpub.com` or e-mail `suggest@packtpub.com`.

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.

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/support`, 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 `http://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.

Instant HTML5 Local Storage How-to

Welcome to *Instant HTML5 Local Storage How-to*, where we will take you on a journey through using Local Storage, using the new HTML5 standard tags, which are now available in most modern browsers.

Basic use of Local Storage (Must know)

When using HTML5 Local Storage, there are two types that you can use—local storage and session storage. We'll begin by looking at the former first, using a simple form as the basis for our recipe.

Getting ready

For this recipe, all you need is your browser and favorite text editor.

How to do it...

Perform the following steps:

1. Let's begin with creating a blank document in your text editor. Then, add the following code and save it as a test `localdemo.html`:

```
<!DOCTYPE html>
<html>
  <head>
    <script src=
    "http://code.jquery.com/jquery-1.8.1.min.js"></script>
    <script type="text/javascript">
```

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

2. In between the `<script>` tags, add the following function. This copes with storing the information within the browser.

```
<script type="text/javascript">
  function storeItem() {
    var item = $('#item').val();

    var items = localStorage.getItem('myItems');
      if (items != null) {
        items = JSON.parse(items);
      } else {
      items = new Array();
      }

    items.push(item);

    localStorage.setItem('myItems',
    JSON.stringify(items));

    refresh();
  }
</script>
```

3. We need to add another function to retrieve information and refresh the content displayed on screen. So go ahead and update the script as highlighted:

```
<script type="text/javascript">
  function storeItem() {
    var item = $('#item').val();
    var items = localStorage.getItem('myItems');
    if (items != null) {
      items = JSON.parse(items);
    } else {
      items = new Array();
    }
    items.push(item);
    localStorage.setItem('myItems',
    JSON.stringify(items));
    refresh();
  }
```

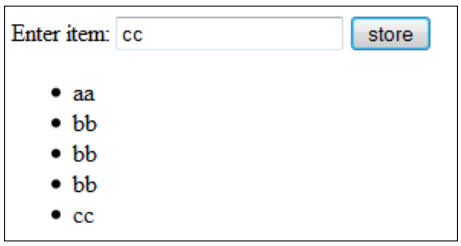
```
  function refresh() {
    var items = localStorage.getItem('myItems');
    var ul = $('ul');
    ul.html('');
    if (items != null) {
      items = JSON.parse(items);
      $(items).each(function (index, data) {
        ul.append('<li>' + data + '</li>');
      });
    }
  }

  $(function () { refresh(); });
</script>
```

4. We finish by adding a basic form—while the purists amongst you will notice that it doesn't have all of the proper forms of tag, it is enough to illustrate how this demo works. Add the following code snippet, just above the closing `</body>` tag:

```
Enter item:
<input type="text" id="item" />
<input type="button" value=
"store" onclick="storeItem()" />
<br />
<ul></ul>
```

5. Crack open your browser and preview the results. Here's a screenshot of what you should see, with some example values already entered:



How it works...

Now we've seen Local Storage in action, let's take a look at how it works in detail.

HTML5 Local Storage works on the principle of named key/value pairs, where you store information using a named key and retrieve it by calling that named key. Everything is stored locally on the user's PC; it cuts down the need to retrieve information from the server, thereby acting as a form of caching.

You may have noticed that we've used jQuery in this recipe—basic use of `LocalStorage` (and `SessionStorage`) doesn't necessarily need jQuery; you could use pure JavaScript if you prefer. It all depends on your requirements; if you are already using jQuery in your pages, for example, you may prefer to use this over JavaScript. (You will see I have used a mix of both throughout this book, to show you how you can use either jQuery or JavaScript).

In this recipe, we've used jQuery to reference `LocalStorage`; if you take a look at the code, you will see two lines of particular importance:

- `var items = localStorage.getItem('myItems');`
- `localStorage.setItem('myItems', JSON.stringify(items));`

These two handle the retrieval and setting of values respectively. In this demo, we begin with either fetching the contents of any existing stored information and inserting them into an array, or creating a new one, if nothing exists within the store. We then use `JSON.stringify()` to convert information from the form into a string, push this into the storage, and then refresh the page so that you can see the updated list. To get the information back, we simply repeat the same steps, but in the reverse way.

The beauty of using JSON as part of storing information in this way is that you are not entirely limited to just plain text; you can store some other things in the `LocalStorage` area, as we will see later in this book.

There's more...

By now, you will start to see that using Local Storage works very much in the same way that cookies do—indeed some people often refer to Local Storage as "cookies on steroids". This said, there are still some limitations that you need to be aware of when using Local Storage, such as the following:

- Local Storage will only support text as a format and is set to a suggested arbitary limit of 5 MB, although this is inconsistent across browsers.
- If you exceed this, the `QUOTA_EXCEEDED_ERR` error is thrown. At the time of writing this book, there is no support built in for requesting more space. Some browsers such as Opera will allow the user to control each site's quota, but this is a purely user-based action.
- Web Storage is no more secure than cookies; although use of the HTTPS protocol can resolve a lot of security issues, it is still up to you as a developer to ensure that sensitive information (such as passwords) is not sent to or stored locally on the client using Web Storage.

With careful use, we can take advantage of the ability to store relevant information on a user's PC, and avoid the need to push it back to the server. Once the information has been stored, there will be occasions when you will need to view the raw information from within your browser—this is easy enough to do, although the method varies from browser to browser, which we will see as part of the next recipe.

Viewing Local Storage content in a browser (Must know)

In the previous recipe, we took a look at how to use the Local Storage functionality in your website. There will be instances where you might want to examine the raw information being stored in your browser—we'll take a look at how to do this within this task.

Getting ready

For this task, you will need at least one modern browser—this can be IE, Chrome, Safari, or Firefox. The version must be a recent one, from within the last two years. For IE users, you will also need access to a working local web server, such as WAMP Server (for PC) or MAMP for Macs. Linux users will likely have one installed, or available to install within the distribution.

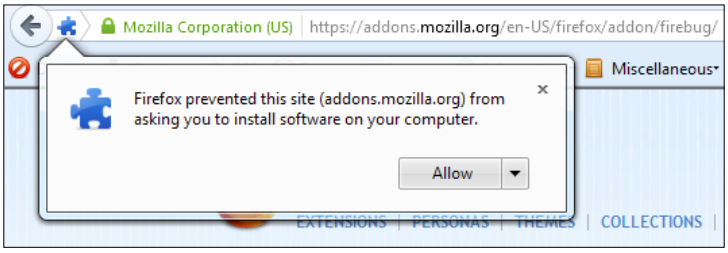
Also, if you are using Firefox, you will need to avail yourself of a copy of Firebug, which you can get from `https://addons.mozilla.org/en-US/firefox/addon/firebug/`.

In this recipe, it is assumed that you have all four browsers installed (including a local web server for IE). You'll see how to view the raw data within each browser.

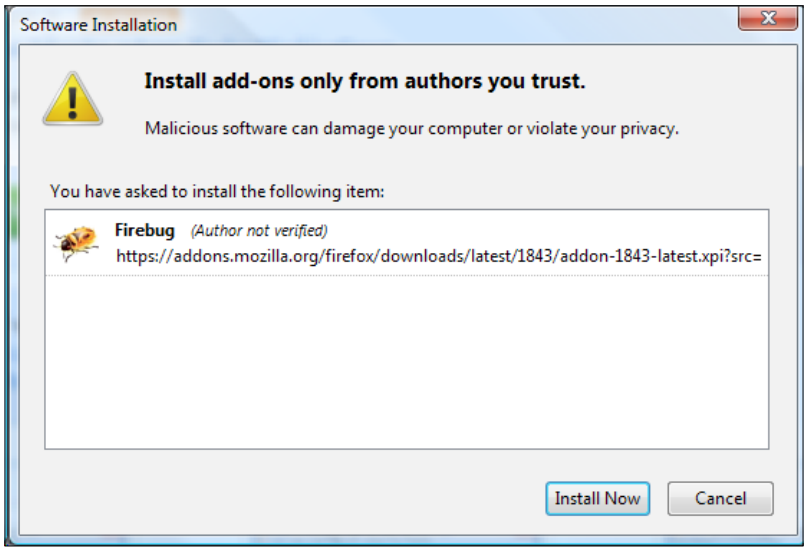
How to do it...

To view the content of Local Storage in a browser, perform the following steps:

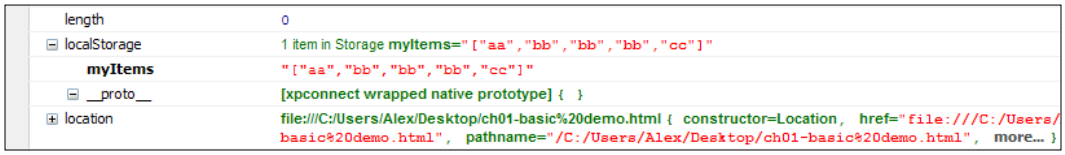
1. Let's start by installing the Firebug plug-in. Visit `https://addons.mozilla.org/en-US/firefox/addon/firebug/` and click on the **Add to Firefox** button. If prompted, click on **Yes** to allow Mozilla to install the plug-in:



2. The plug-in will download and prompt you to install it:



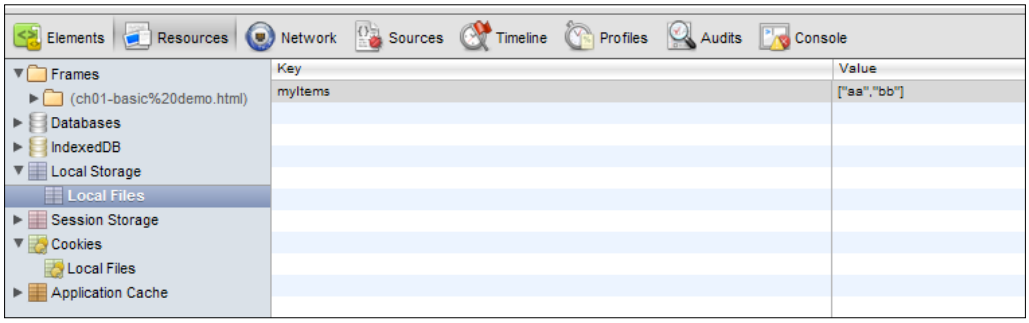
3. We don't need to restart Firefox to complete the installation for Firebug, so you will now be able to view the elements stored in Local Storage. The raw content is stored in the `webappsstore.sqlite` file within the `profile` folder. You can't edit it manually there, but can do so by pressing *F12* in Firefox to bring up Firebug. Click on **DOM** and then **Show DOM Properties** to show the values being stored in the Local Storage database:



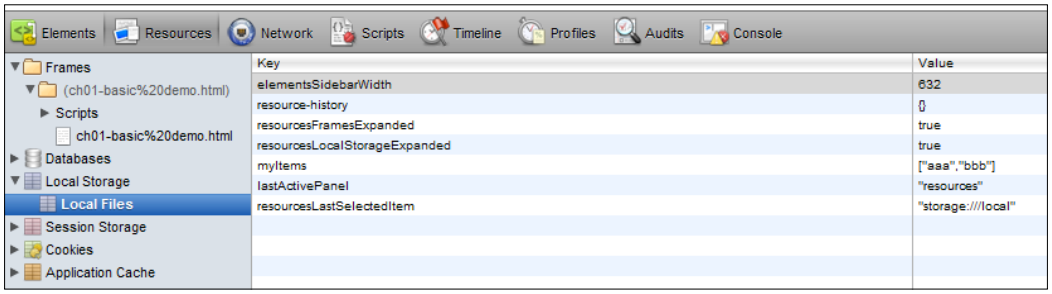
Downloading the example code

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.

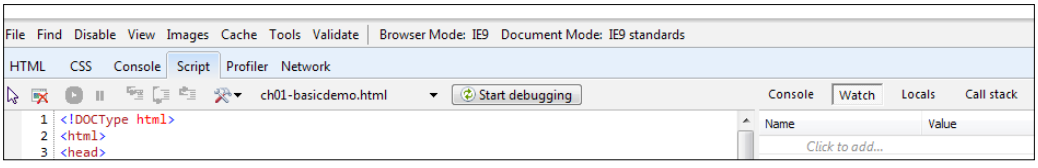
4. Let's focus our attention now on Chrome. The support for viewing the Local Storage content is already built-in by default, so go ahead and fire up Chrome. Then, right-click anywhere on the page and select **Inspect Element** to bring up the Developer toolbar. Select **Resources** to view the content. You will see an entry for Local Storage (which has two values showing, in this example):



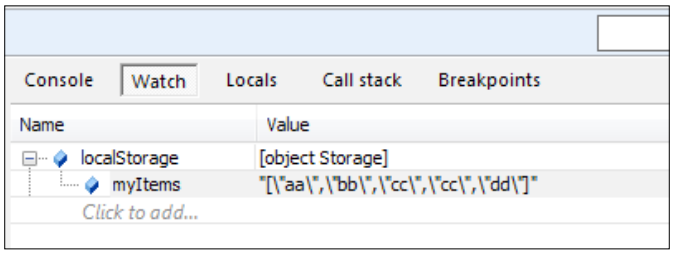
5. Next in our list of browsers is Safari. This works in the same manner, so go ahead and open up your copy of Safari, right-click anywhere on the page, and select **Inspect Element**. Then, click on the **Resources** tab to see the Local Storage entries:



6. Last, but by no means least, is Internet Explorer. This is a little more involved, so save a copy of your demo file within your web server's `www` folder on your PC. Ensure your web server is started. Then, open up Internet Explorer and browse to that file from within your web server.
7. In IE, press *F12* to bring up the Developer toolbar and then switch to the **Script** tab:



8. Click on **Click to add**, then enter `localStorage`, and press *Enter*. Note that this is case sensitive! You will see the contents of `localStorage`:



You may prefer to use one of the tools available to view the `localStorage` contents in IE. Have a look at Foundstone HTML5 Local Storage Explorer 1.1, which is available as an add-in from `https://addons.mozilla.org/en-US/firefox/addon/foundstone-html5-local-storage/`.

How it works...

In this recipe, we looked at the raw information that the browser holds, when asked to store content within its `localStorage` area. While the means to view it might differ from browser to browser, the principles are still the same. Each browser will reference the contents of the `localStorage` area on your client PC in similar ways. This content is stored at various locations:

- In IE 8 or higher, content is stored as an XML file in `%userprofiles%/Local Settings/Application Data/Microsoft/Internet Explorer/DOMStore`.
- For Firefox users, data is stored in the `webappsstore.sqlite` file in the `profile` folder and you can also get hold of some add-ons to view the contents.
- If you're using Chrome, this is a little more involved, depending on the operating system being used, for example, in Windows XP, the content is stored in `C:\Documents and Settings\%username%\Local Settings\%Application Data%\Google\Chrome\User Data\Default`.

  If you are using Vista or Windows 7, it's in `C:\Users\%username%\%AppData%\Local\Google\Chrome\User Data\Default`.
- Last, but not least, content for Safari is stored in the `sqllite` file in `C:\Users\%username%\AppData\Local\Apple Computer\Safari\LocalStorage`. There's a Local Storage file named the same as the website storing it.

There's more...

If you want to get really down and dirty in your browser, you can use the console application that comes as part of each browser. It is possible to get and set values into Local or Session Storage, using the command line, in much the same way that you would do using something like JavaScript. The beauty of this is that it doesn't need any additional plugins—the exception being IE, which needs its Developer toolbar, but this is included by default anyway!

As an example, here's how you would do this if you were using Firefox. Navigate to the domain in question and open Firefox's console (*Ctrl* + *Shift* + *K* in Windows). You can then enter any of the following actions as shown in the examples:

- **Set a value**: `localStorage.foo = "bar"` or use `localStorage.setItem ("foo", "bar")`
- **Change a value**: `localStorage.setItem ("foo", "all day")` or use `localStorage.foo = "all day"`
- **Delete a value**: `localStorage.removeItem ("foo")`

This is not the easiest way to do it, but it still works, just in case you need it.

We're going to move on and look at the second type of storage available, which is Session Storage. Although it is similar to Local Storage, there are some noticeable differences, as you will see in the next recipe.

Basic demo for Session Storage (Must know)

In this recipe, we're going to take a look at how Session Storage works, using a simple demo in the form of a click counter, which will reset each time you refresh your browser session.

Getting ready

For the purposes of this task, all you will need is a normal text editor of your choice.

How to do it...

In the following steps, we will take a look at how Session Storage works:

1. Let's start by creating a new file. Add the following framework for our demo. Save it as `sessiondemo.html`:

```
<!DOCTYPE html>
<html>
  <head>
    <meta content="utf-8">
```

```
    <script src=
    "http://code.jquery.com/jquery-1.8.1.min.js"></script>
    <script type="text/javascript"></script>
  </head>
  <body>
  </body>
</html>
```

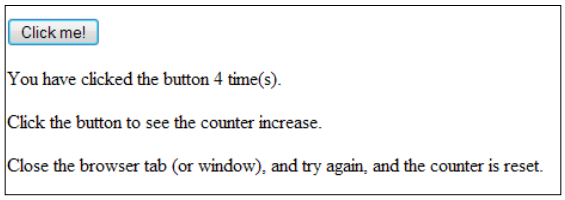
2. We need to add a button and some text. We'll use the following code snippet to illustrate how Session Storage works:

```
<p><button id="clickCounter" type="button">Click me!</button></p>
<div id="result"></div>
<p>Click the button to see the counter increase.</p>
<p>Close the browser tab (or window), and try again, and the
counter is reset.</p>
```

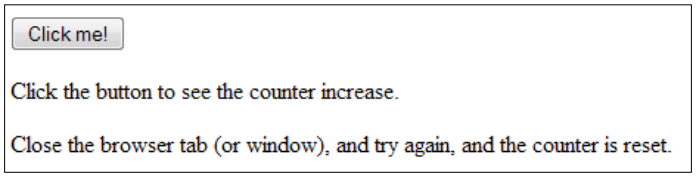
3. Finally here comes the real meat. Add the following in between your `<script>` tags, which will keep a count of how many times you click on the button and display the results on the screen:

```
$(document).ready(function(){
  $("#clickCounter").click(function () {
    if (sessionStorage.clickcount) {
      sessionStorage.clickcount=Number
      (sessionStorage.clickcount)+1;
    } else {
      sessionStorage.clickcount=1;
    }
    $("#result").html("You have clicked the button " +
    sessionStorage.clickcount + " time(s).");
  });
});
```

4. If all is well, you will see the following result in your browser. Here, the button has already been clicked four times:



5. Close your browser. What do you see? If all is well, you should see the result as shown in the following screenshot, as the session information being held has been reset by the browser:



How it works...

In this recipe, we've set up a simple HTML document, which has a button that acts as a counter. We start with a check to ensure that there is a `clickcount` property in Session Storage. If not, we create a new one and assign it the value `1`. Each time we click on the button, it increases the `clickcount` attribute of Session Storage by a factor of one and updates the count on the screen.

There's more...

The final step in this recipe is arguably the most important one in the whole recipe. It illustrates perfectly that Session Storage only lasts for as long as the current browser session. It is sufficient to merely refresh the screen. As soon as you do that, you will lose the count, which the browser resets back to `1` in preparation for a new counting session. If your data needs to be more persistent, you will need to use Local Storage instead, as we will see in the next recipe.

Comparing Local Storage and Session Storage (Should know)

At the beginning of this book, we took a look at how you can use Local Storage to store persistent information on a client computer, which remains there even when the browser is closed. That's great, but isn't it a little overkill for instances such as shopping carts? You'd be right – we don't need to store information about purchases forever; indeed, information should only be stored for the duration of the current session. In this recipe, we're going to take another look at Session Storage, to see how it stacks up against LocalStorage.

How to do it...

In the following steps, we will take a look at how it works and stacks up against Local Storage:

1. Let's begin by creating a new HTML document. Add the following code snippet and save it as `sessionvlocaldemo.html`:

```
<!DOCType html>
<html>
<head>
</head>
<body>
  <div id="content">
</div>
</body>
</html>
```

2. This code needs to use jQuery in order to work, so let's go ahead and add it. We want to make sure the results on screen look presentable, so we'll add in some basic styling for good measure:

```
<script src="http://ajax.aspnetcdn.com/ajax/jQuery/jquery-
1.8.0.min.js"></script>
<style type="text/css">
  #content { width: 400px; padding: 10px;
  margin-left: auto; margin-right: auto;
  margin-top: 5%; border: 1px solid #c4c4c4; }
</style>
```

3. We need to add our script, which will manage the Session Storage requests. Begin with the event handler that manages how many times we click on the request button:

```
  <script type="text/javascript">
  function clickCounter()
  {
    if(typeof(Storage)!=="undefined") {
    if (sessionStorage.clickcount) {
      sessionStorage.clickcount=Number
      (sessionStorage.clickcount)+1;
    } else {
      sessionStorage.clickcount=1;
    }
    document.getElementById("result").innerHTML=
    "You have clicked the button " +
    sessionStorage.clickcount + " time(s).";
    } else {
      document.getElementById("result").innerHTML=
```

```
      "Sorry, your browser does not support web
      storage...";
    }
  }
```

4. The following function gets the requested data and pushes it into the `localStorage` area:

```
function storeItem() {
  var item = $('#item').val();
  var items = localStorage.getItem('myItems');
  if (items != null) {
    items = JSON.parse(items);
  } else {
    items = new Array();
  }
  items.push(item);
  localStorage.setItem('myItems', JSON.stringify(items));
  refresh();
}
```

5. Then, we need to refresh the screen to update values on the screen:

```
function refresh() {
  var items = localStorage.getItem('myItems');
  var ul = $('ul');
  ul.html('');
  if (items != null) {
    items = JSON.parse(items);
    $(items).each(function (index, data) {
      ul.append('<li>' + data + '</li>');
    });
  }
}
```

6. The following script calls the `refresh` function that we've just set up:

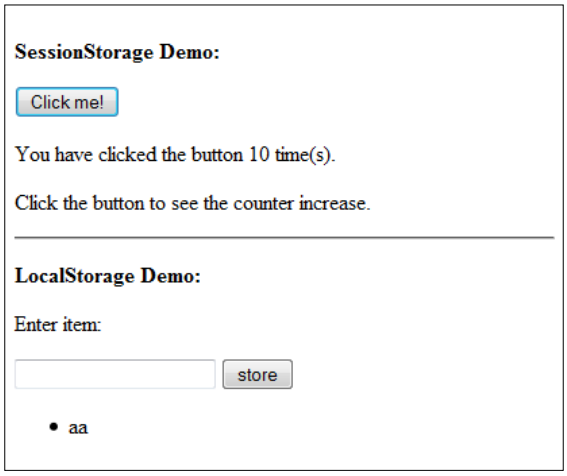
```
$(function () {
  refresh();
});
</script>
```

7. Finally, we need to provide the means to request data from the user for both Session Storage and Local Storage:

```
<p><b>SessionStorage Demo:</b></p>
<p><button onclick="clickCounter()" type="button">
Click me!</button></p>
```

```
<div id="result"></div>
<p>Click the button to see the counter increase.</p>
<hr>
<p><b>LocalStorage Demo:</b></p>
<p>Enter item:</p>
<input type="text" id="item" />
<input type="button" value="store" onclick=
"storeItem()" />
<br />
<ul></ul>
```

8. You should see something similar to the following screenshot, when previewing in a browser:



How it works...

At the heart of Session Storage, there is actually very little difference compared to that of Local Storage. Both work on the same basis of setting and getting a named key/value pair within the browser's storage area. Clicking on the `SessionStorage` button will increase the counter `ad` to infinitum. It will be reset as soon as you refresh your browser window. In contrast, adding any entries to the `LocalStorage` option will store the values; these will remain stored, irrespective of any browser refresh.

If you need information to remain on the browser after it has been closed or restarted, you should use Local Storage. If, however, you're only looking to keep values for the duration of a single session, Session Storage is the better option.

There's more...

Now that we have had an opportunity to look at Session Storage, let's dig into how both methods work in more detail.

Both Local and Session Storage work using the same methods:

Type	Name and arguments	Purpose
string	`key(int)`	Returns the name of a key at the index in the argument
string	`getItem(string key)`	Returns the value of the key
void	`setItem(string key, string data)`	Assigns a value to a key
void	`removeItem(string key)`	Removes the key whose name is in the argument
void	`clear()`	Clears the storage space and removes all keys
long	`length()`	Number of items stored

The Storage API has been defined so that each of the `get`, `set`, and `remove` methods are known as getters, setters, and deleters. To see what I mean, take a look at the following two examples:

```
var myVar = sessionStorage.getItem('myKey');
sessionStorage.setItem('myKey', 'myValue');
```

These two methods get and set the information into the `myKey` value within the `localStorage` area of the browser as a `sessionStorage` key/value pair. However, you could equally write the following:

```
sessionStorage.myKey = 'myValue';
var myVar = sessionStorage.myKey;
```

The preceding code would achieve the same result and is probably a little more readable too! The same applies to `localStorage`:

```
localStorage.setItem('myKey', 'myValue');
var myVar = localStorage.getItem('myKey');
```

You could write this as follows:

```
localStorage.myKey = 'myValue';
var myVar = localStorage.myKey;
```

The preceding code will achieve the same result.

Now we've seen how Session Storage works. We will now see some more differences between Session Storage and Local Storage. Before doing so, we're going to take a look at a theoretical example of how you could approximate the effect of Session Storage if you don't have any access to jQuery or a library such as Modernizr, or to a more recent browser. It will look ugly, but there is a reason for showing you this. All will become clear as we go through the next recipe.

Using web storage instead of cookies (Must know)

In this recipe, we're going to take a look at how you can approximate the effect of Session Storage, if you are forced use really old browsers. We'll first look at a simplified example of how you can use Local Storage and compare that to the closest available effect, if you had to use just cookies.

Please note this recipe is intended to only serve as an example. It is not one that I would recommend using unless you need to maintain support for very old browsers, which is very unlikely. Most recent browsers will support the first part of this demo without too much of an issue.

How to do it...

To use web storage instead of cookies, perform the following steps:

1. Crack open a new HTML document and add the following code snippet. Then, save it as `cookieswap.html`:

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content=
    "text/html; charset=utf-8">
    <title>Untitled 1</title>
    <script type="text/javascript">
    </script>
```

```
  </head>
  <body>
  </body>
</html>
```

2. We need to add a script that calls `localStorage`, so add the following code snippet between the `<script>` tags in the header of our document:

```
<script type="text/javascript">
// if localStorage is present, use that
if (('localStorage' in window) && window.localStorage !== null) {
  // easy object property API
  localStorage.wishlist =
  '["Jaguar","Bugatti","Aston Martin"]';
} else {
```

3. The second part of this script caters for a fall back in the event that `sessionStorage` isn't available at all. We can achieve a similar effect using the `document.cookie's` API:

```
  // without sessionStorage we'll have to use
  a far-future cookie
  //   with document.cookie's awkward API :(
  var date = new Date();
  date.setTime(date.getTime()+(365*24*60*60*1000));
  var expires = date.toGMTString();
  var cookiestr = 'wishlist=["Jaguar","Bugatti",
  "Aston Martin"];' + ' expires='+expires+'; path=/';
  document.cookie = cookiestr;
}</script>
```

How it works...

In this recipe, it is easy to see why Local Storage becomes very useful. Without it, we are limited to creating a very crude effect of Session Storage! Here, we've used the `document.cookie's` API to set up a cookie that is set to a date at some distance in the future, to prevent deletion of the cookie. We then create a variable that concatenates together the information we want to store and the date that needs to be set. This is assigned to `document.cookie`, thereby creating our cookie. If we use this code, when the date has been set sufficiently far ahead, the cookie will not be deleted.

There's more...

The danger in using this method is that users can easily delete cookies through their browsers. One could argue that it is easy enough to do the same to the elements stored in Local Storage, although the latter requires a few more steps than simply deleting cookies through a browser's GUI. However, it does not alleviate the fact that cookies are restricted to specific domains and browser instances (that is, specific tabs in the browsers). Local Storage breaks these constraints by making content available to all tab instances in a browser, as well as allowing use of more types of content, provided they can be converted to string format as we will see later in this book.

Basic detection for storage support (Must know)

Great! We've looked at Session Storage and Local Storage, and learnt how we can use both to store information in the browser rather than having to push it back to the server.

There's one thing wrong with the preceding statement. The more observant amongst you would have noted from my earlier comments that while browser support is actually very good, it still isn't supported in older browsers!

After all, it is a part of the HTML5 specification, which has naturally only been available over the last few years, and has garnered support from the more recent browsers. While we can't retrofit support to older browsers, we can at least implement a basic check for it, which is the subject of our next task.

Getting ready

This is a simple recipe for which you will need a copy of the code from the *Basic demo of Session Storage* recipe discussed earlier in this book, along with a normal text editor of your choice.

Although we're going to look at the support for Session Storage in this recipe, the principles are the same for Local Storage and can be used in both forms with no issue.

How to do it...

The following steps illustrate how to perform a basic detection for storage support:

1. Let's make a start by opening a copy of the `sessionStorage` demo, which we created earlier in this chapter. The first part of the code remains as it is. Alter the next part of the code as follows:

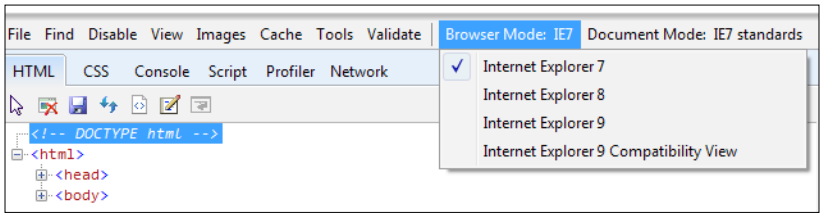
```
  $("#clickCounter").click(function () {
    if (typeof(Storage)!=="undefined") {
      if (sessionStorage.clickcount) {
        sessionStorage.clickcount=Number
        (sessionStorage.clickcount)+1;
      } else {
        sessionStorage.clickcount=1;
      }
      $("#result").html("You have clicked the button " +
      sessionStorage.clickcount + " time(s).");
    } else {
      $("#result").html("Sorry, your browser does not
      support web storage...");
    }
  });
});
</script>
```

2. The rest of the code remains as it is, from the previous task:

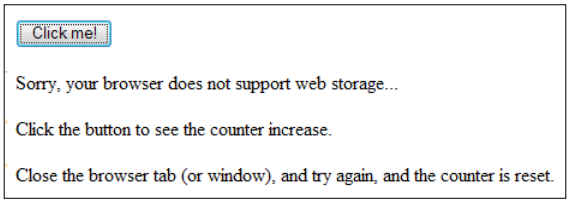
```
  </head>
  <body>
    <p><button id="clickCounter" type="button">
    Click me!</button></p>
    <div id="result"></div>
    <p>Click the button to see the counter increase.</p>
    <p>Close the browser tab (or window), and try again,
    and the counter is reset.</p>
  </body>
</html>
```

3. To show this in action, save your changes to a new file called `ssnotsupported.html` within the `www` folder of your local web server.

4. Fire up IE, then activate IE's Developer toolbar, and change the user agent to mimic IE 7:



5. Browse to your page, and then try clicking on the button to start the count. You will see that it fails, as although we are using IE 9, the change of user agent forces IE to think as it is IE 7; Location Storage isn't supported in IE 7:



How it works...

Although the change we've made is a simple one, it is critical for the successful use of Local Storage. We still live in an age when older browsers such as IE 6 have to be used. We've adapted the `sessionStorage` code from a previous task to perform a simple JavaScript-based check to ensure that the code only runs if it is supported in the browser being used. In this instance, although we tested the code in IE 9, we forced it to act as if it was IE 7. The check correctly identified this and displayed the appropriate message.

This does beg one question, is it really necessary to do this? The answer is unfortunately yes, at least until older browsers such as IE 6 have been put out to pasture. Microsoft's website (`http://www.iecountdown.com`) currently has this true for 6 percent of users worldwide. Once this drops below 1 percent, we can comfortably discard the need to support these browsers!

Using JavaScript's `typeof()` keyword forces you to manually provide every eventuality. A better option would be to use the power of Modernizr to provide CSS hooks that you can then style to your heart's content. Modernizr supports checking for a wide variety of functionality, based on what the browser can handle, and not its user agent string (which can be faked and therefore be unreliable). We're going to take a look at how you can use it to better detect support for Local and Session Storage, as part of the next recipe.

Improving detection using Modernizr (Should know)

In the previous recipe, we used some basic JavaScript to perform a rudimentary check to see if the Local Storage or Session Storage support is enabled. Since this works, it is not infallible. As part of this recipe, we will use Modernizr to perform a check that is more reliable, and which works better with the advent of HTML5 functionality.

Getting ready

For this recipe, we will need some icons—one cross and one checkmark—in addition to a normal text editor of your choice. A good source is the silk icons available from FAMFAMFAM at `http://www.famfamfam.com/lab/icons/silk/`. For the purposes of this recipe, I will assume you've downloaded and are using the icons from this library.

How to do it...

For improving detection using Modernizr, perform the following steps:

1. Let's begin by creating our base template. Save the following code snippet as a new HTML document, calling it `modernizr.html`:

```
<!DOCTYPE html>
<html class="no-js" lang="en">
  <head>
    <meta content="charset=utf-8">
    <script src="http://code.jquery.com/jquery-latest.js">
    </script>
    <script src="http://www.modernizr.com/downloads/
    modernizr-latest.js"></script>
    <style type="text/css">
    </style>
  </head>
  </head>
  <body>
  </body>
</html>
```

2. Next comes the base dialog message, which we will use to confirm if support for Local Storage or Session Storage is available:

```
<body>
  <div id="supportdlg">
    <div id="details">
      <div id="localtxt">LocalStorage is not supported or
      javascript is disabled</div>
      <div id="sessiontxt">SessionStorage is not supported
      or javascript is disabled</div>
    </div>
  </div>
</body>
```

3. We need to add some styling, which includes the fallback properties, in the event that Modernizr is not available:

```
<style type="text/css">
  body { padding: 20px; font-family: Arial;
  font-size: 16px; }
  #localtxt { margin: 5px; }
  #sessiontxt { margin: 5px; }
  #details { border-radius: 4px; padding: 5px;}
  supported { background-image:url('tick.png');
  background-repeat: no-repeat; padding-left: 25px;
  background-color: green; }
  notsupported { background-image:url('cross.png');
  background-repeat: no-repeat; padding-left: 25px;
  background-color: red; }
  #supportdlg { border-radius: 4px; border: 1px solid
  black; padding: 5px; width: 500px; color: #fff;
  font-weight: bold; }
  no-js #supportdlg { background-color: red;
  border: 1px solid black; }
  /* No JavaScript Fallback for Modernizr */
</style>
```

4. Finally, we need to provide the glue that will perform the check for the support. We begin with the check for Local Storage, which displays the result on the screen:

```
<script>
  if(Modernizr.localstorage){
    $('#localtxt').html("LocalStorage is supported");
    $('#localtxt').removeClass().addClass("supported");
    $('#details').css("background-color", "green");
  } else {
    $('#localtxt').html("LocalStorage is not supported");
    $('#localtxt').removeClass().addClass("notsupported");
    $('#details').css("background-color", "red");
  }
```

5. We then add the check for `sessionStorage`, to confirm if our browser supports Session Storage. It displays the appropriate result based on the outcome:

```
  if(Modernizr.sessionstorage){
    $('#sessiontxt').html("SessionStorage is suppported");
    $('#sessiontxt').removeClass().addClass("supported");
    $('#details').css("background-color", "green");
  } else {
    $('#sessiontxt').html("SessionStorage is not
    suppported");
```

```
        $('#sessiontxt').removeClass().
        addClass("notsupported");
        $('#details').css("background-color", "red");
    }
</script>
```

6. If all is well, you will see the result as shown in the following screenshot, when previewing it in a browser:

How it works...

We kick off this recipe by hooking in Modernizr and jQuery. Then add some styles that will be used as part of displaying the results of the status check. We also include a base set of DIVs that will always show if both JavaScript and Modernizr are not available.

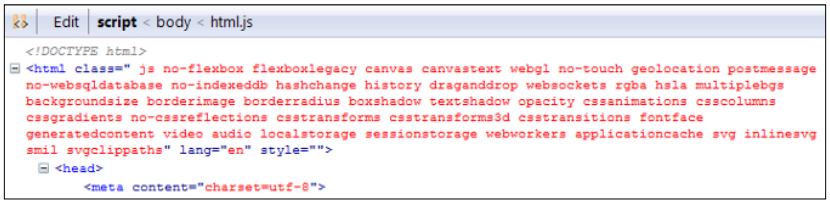
The key to this recipe is in the first line of the script:

```
if(Modernizr.localstorage){
```

This performs a check using Modernizr, to confirm if a browser is capable of supporting a feature, such as Local Storage or Session Storage. If it is, we alter the styles and text assigned to the `localtxt` and `sessiontxt` DIVs accordingly.

If you are interested in exploring Modernizr in more depth, you should take a look at the book *Learning HTML5 Modernizr* by *Adam Watson*, *Packt Publishing*.

The beauty of using Modernizr in instances such as these is its ability to automatically add styling hooks that you can use to cater for instances when a feature is or isn't supported. It shuns the user agent sniffing technique that was popular when the Internet first came into (mainstream) existence, in preference to working out if a browser is able to support a particular feature. It adds a number of style rules in the header and alters any that relate to unsupported technologies:



You can then use these styles accordingly, depending on what is provided by Modernizr.

There's more...

If you want to confirm that Modernizr is indeed working, you can switch off DOM Storage in your browser. The preceding code will display the following result:

If, however, JavaScript is not even enabled, you will see the following result:

The means to perform this will vary from browser to browser. Have a look online for exact instructions on how to alter these settings. For example, in Firefox, you can switch off DOM Storage by toggling `dom.storage.enabled` in the `about:config` page. Please do this at your own risk!

A small thing to note is that while Modernizr makes providing fallback support very easy, you should always ask yourself the question, "if I'm using Modernizr, should I be using Modernizr?" Modernizr can add some overhead, as it always loads both normal and fallback content, scripts, and styles; you can reduce the impact of this by using the conditional loader script `Yesnope.js`, which comes bundled with Modernizr, or is available standalone from `http://www.yesnope.com`. It may not make a difference in a small example such as ours, but will have a significant impact on much larger sites!

Now, it's good talking about checking for whether a browser supports Local Storage, but what happens if it doesn't? Thankfully, it is easy to fix. Let's take a look at `store.js`, which provides a good fallback for those browsers that don't support Local Storage natively, as part of the next recipe.

Providing fallback support (Should know)

In this recipe, we're going to take a look at how you can provide the fallback support, using the `store.js` library.

Getting ready

For this recipe, we'll need a copy of the open source `store.js` library, which is available from `https://github.com/marcuswestin/store.js/archive/master.zip`. We will also need a working installation of WAMP Server, or an available remote site, where we can upload the code. I will assume you're using WAMP Server, for the purposes of the demo. We will, of course, also need our trusty text editor!

How to do it...

Perform the following steps. for providing the fallback support:

1. Let's begin with adding the following code to a new HTML document. Then, save it as `fallback.html`:

```
<!DOCTYPE html>
<html>
  <head>
  </head>
  <body>
    <div id="storeout">
      <div id="details"></div>
    </div>
  </body>
</html>
```

2. Go ahead and add the link into `store+json2.min.js` and some decorative CSS styles into your `<head>` tags:

```
  <script src="store+json2.min.js"></script>
  <style type="text/css">
    #storeout { border: 1px solid #000; padding: 5px;
    margin-left: 5px; margin-top: 5px; width: 450px; }
    #details { background-color: #b80000;
    font-weight: bold; font-family: Arial; color: #fff; }
    #details p { width: 325px; padding: 5px;}
  </style>
```

3. We need to add a check to ensure whether `LocalStorage` as a fallback will work or not; if it does, we then set some values into `LocalStorage`, using `store2.js`:

```
<script>
  if (store.enabled) {
    document.write("<p>LocalStorage is available!</p>");
    store.set('myage', 38);
    store.set('user', { name: 'Alex', likes: 'jquery' })
```

```
        document.write("<p>Hi my name is " +
        store.get('user').name + " and I'm " +
        store.get('myage') + " years old.</p>");
      } else {
        alert("Sorry, localStorage is not available.");
      }
    </script>
```

4. If all is well, you should see the following when previewing `LocalStorage` in a browser:

How it works...

`Store.js` has been designed to work in a similar manner to `LocalStorage`, to the extent that it can almost be dropped in as a replacement for `LocalStorage`; it only requires some minor changes in the existing code to work.

We kick off our recipe with a call to the library. You will note that it includes a reference to `json2`; this is because `LocalStorage` expects to call a `toString` method on all information it is asked to store. This will prevent you from storing numbers, objects, and arrays. To get around this, we use JSON to convert the object to string before storing it. We then move into the main script, which sets a key called `myage`, into which it stores a numeric value; we also create an array called `user`, into which it stores my name and that I like jQuery! We then make references to both from the next call, as shown in the screenshot.

So far, we've now looked at how you can detect and provide support for most browsers; I am sure there will be one question you will want answered though, "what about the mobile platform?" It's a good question, one we can easily fix by looking at how Local Storage can be used on the mobile platform, as part of the next recipe.

Is the user online or offline? (Become an expert)

So far, we've looked at how you can use Local Storage (and Session Storage) in your projects, as well as determined if your browser even supports the technology. However, this is all good and well if you're using a normal PC, but what about anyone using a mobile platform, such as iPads or smartphones? We need to know if the unit is working offline, particularly in cases of poor reception. Thankfully, it is really easy to fix this, using our good friend Modernizr.

Getting ready

For this recipe, all you will need (in addition to your normal text editor) is two icons—if you have them to hand. The two we used in the *Improving detection using Modernizr (Should know)* recipe discussed earlier in this book will be perfect.

How to do it...

Perform the following steps, to check whether the user is online or offline:

1. Ok, let's make a start. Copy the following code into a new file and save it as `offline.html`:

```
<!DOCTYPE html>
<html class="no-js" lang="en">
  <head>
    <meta content="charset=utf-8">
  </head>
  </head>
  <body>
    <div id="supportdlg">
      <div id="details">
        <div id="online">Website is offline or JavaScript
        is not available</div>
      </div>
    </div>
  </body>
</html>
```

2. We need to initialize two references, one to jQuery and another to Modernizr. So add the following code between the `<head>` tags in your code:

```
<script src="http://code.jquery.com/jquery-latest.js"></script>
<script src="http://www.modernizr.com/downloads/modernizr-latest.
js"></script>
```

3. Next comes the meat of the demo. This is the script call to determine if your browser is online or offline:

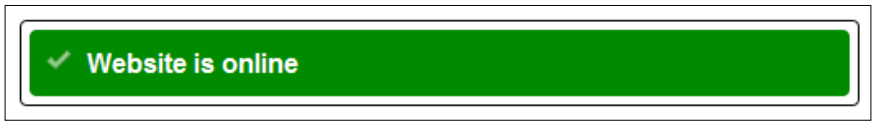
```
<script>
    if(Modernizr.applicationcache) {
      $('#online').html("Website is online");
    }
</script>
```

4. Finally, we need to add some styling, as the preceding code certainly won't win any style awards:

```
<style type="text/css">
  body { padding: 20px; font-family: Arial;
  font-size: 16px; }
  #details { border-radius: 4px; padding: 5px;
  background-color: green;}
  #supportdlg { border-radius: 4px; border:
  1px solid black; padding: 5px; width: 500px;
  color: #fff; font-weight: bold; }
  #online { background-image:url('tick.png');
  background-repeat: no-repeat; padding-left:
  25px; background-color: green; margin: 5px; }

  /* No JavaScript Fallback for Modernizr */
  .no-js #details { background-color: red;}
  .no-js #online { background-image:url('cross.png');
  background-color: red; }
</style>
```

5. So, if all is well, you should see either of the following results when previewing in your browser. The first one you will see straightaway:



6. The following result will be shown if you select your browser's offline function or disconnect from the Internet:

How it works...

This recipe works on a simple principle of replacement, but only when JavaScript is available and the browser has not been set to work offline. We set up a basic HTML framework (simplified for the purposes of this recipe), which could easily form the basis for a more complicated project. We then include calls to jQuery and Modernizr, and use Modernizr to ask the browser, "Can I do application cache?" If the answer is yes, we then replace the contents of the `#online` DIV with appropriate text and styling, otherwise leave it untouched.

There's more...

When using Modernizr, the normal approach would be to control what happens when a test is proved positive or negative. The drawback of this is a reliance on JavaScript to provide the answer. This may not be ideal if JavaScript is switched off (which of course we're testing anyway).

A better option is to incorporate the fallback into the base code (as we have done here), and then use Modernizr to switch in what should be the normal code to use. This means that if JavaScript is unavailable, it will always show the fallback code, with no reliance on JavaScript to provide the answer! In a sense, this progressive replacement technique is similar to progressive enhancement for CSS, but uses JavaScript in this instance.

Let's take this concept a little further. Now that we can tell if a device is offline or not, what if we could actually still use our website or application while it is offline? Well, you can, with the use of a cache manifest, which is the subject of our next recipe.

Using a manifest for caching (Become an expert)

In the previous recipe, we looked at how you can know if your mobile device is offline—let's take this a step forward by telling your browser how to cache content using a manifest, in the event it is unable to connect to the Internet.

Getting ready

For this recipe, all you will need is your normal text editor, as well as a working installation of a local web server, such as WAMP (for PC) or MAMP (for Macs). For the purposes of this recipe, I will assume you are using WAMP. You will also need a copy of the code that is available with this book; it contains some of the files required for this task.

How to do it...

Perform the following steps, to use a manifest for caching:

1. Let's make a start by downloading and extracting the `manifest.html` file from the code available with this book. Save a copy of the file into a folder on your PC.
2. We need to create a manifest file. In the code available with this book, look for a file called `cache.appcache`, and save a copy in the same folder as `manifest.html`. Repeat the same process, but this time with a file called `offline.html`, which is available in the code download that accompanies this book.

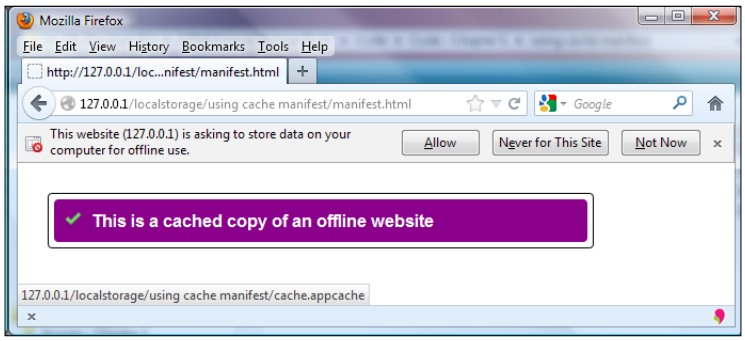
3. Our main file, `manifest.html`, will look a little ordinary. Let's fix that by adding in some styling:

```
body { padding: 20px; font-family: Arial; font-size: 16px; }
#details { border-radius: 4px; padding: 5px; background-color:
purple;}
#supportdlg { border-radius: 4px; border: 1px solid black;
padding: 5px; width: 500px; color: #fff; font-weight: bold; }
#online { background-image:url('tick.png'); background-repeat: no-
repeat; padding-left: 25px; background-color: purple; margin: 5px;
}
```

4. To make the manifest work, we need to alter WAMP. Add the following code at line 412, in `C:\wamp\bin\apache\apache2.2.22\conf`:

```
    AddType text/cache-manifest .appcache
</IfModule>
```

5. If all is well, you should see something similar to the result shown in the following screenshot, when previewing the manifest in your browser:



How it works...

In this recipe, we're using a manifest file to tell which content must be made available when our users are offline. Your browser will prompt you to confirm the acceptance that content can be stored offline, as outlined in the manifest file. Each file consists of three sections, not all of which are required. These sections are `CACHE`, `NETWORK`, and `FALLBACK`. `CACHE` contains those items that should be made available offline. `NETWORK` contains those items that must always be referenced from the server, even when offline. `FALLBACK` provides those items that will be used if the `CACHE` items are unavailable.

Using cache manifest files can be very powerful, but tricky to debug; it is worth reading about them online to get a real feel of how they work, and how you can avoid some of the quirks that are associated with using cache files.

Once the site has been cached, we can confirm if the manifest has worked by viewing the cache details, as shown earlier in this book.

Now that we've covered some of the more basic aspects of Local Storage, it's time to move on and look at some more advanced aspects.In the next recipe, we'll take a look at how you can incorporate the Local Storage support into a website for the mobile platform.

Providing support for the mobile platform (Should know)

So far we've concentrated on how we can use Local Storage within the confines of our site. However, more and more people are using the mobile platform as a means of surfing the web—this isn't a problem as Local Storage is supported in the mobile arena. We're going to have a look at how we can use it as part of this recipe.

Getting ready

For this recipe, you will need a few things in addition to your normal text editor:

- jQuery Mobile CSS: `http://code.jquery.com/mobile/1.2.0/jquery.mobile-1.2.0.min.css`
- jQuery Mobile: `http://code.jquery.com/mobile/1.2.0/jquery.mobile-1.2.0.min.js`
- jQuery: `http://code.jquery.com/jquery-1.8.3.min.js`
- Modernizr: `http://www.modernizr.com/downloads/modernizr-latest.js`

I will assume you have saved local copies of the above libraries for this recipe. If you prefer, you can use the CDN equivalents; you will need to alter the code accordingly.

How to do it...

To provide support for the mobile platform, perform the following steps:

1. Crack open your text editor, and save the following code as `mobilelocalstorage.html`:

```
<!DOCTYPE html>
<html>
  <head>
    <title>jQuery Mobile local storage demo</title>
    <meta name="viewport" content="width=device-width,
    initial-scale=1">
    <link rel="stylesheet"
```

```
    href="jquery.mobile-1.2.0.min.css" />
    <script src="jquery-1.8.3.min.js"
    type="text/javascript"> </script>
    <script src="jquery.mobile-1.2.0.min.js"
    type="text/javascript"></script>
    <script src="modernizr.js"
    type="text/javascript"></script>
  </head>
  <body>
  </body>
</html>
```

2. We need to provide an interface where we can add items to the `LocalStorage` area. So go ahead and add the following code snippet in between the `<body>` tags:

```
<section data-role="page" id="LocalStorageDemo">
  <section data-role="header">
    <h2>jQuery Mobile local storage demo</h2>
  </section>
  <section data-role="content">
    <p id="message"/>
    <ul data-role="listview" data-inset="true">
      <li data-role="list-divider">Enter text to store</li>
      <li><input type="text" id="entry" name="entry"
      placeholder="Enter text to store"/>
      <input type="button" id="addToStorage"
      value="Add to local storage"/></li>
      <li data-role="list-divider">Item in store</li>
      <li class="storeItem"/>
    </ul>
  </section>
</section>
```

3. For the page to work correctly, we need to make a couple of tweaks to the styling:

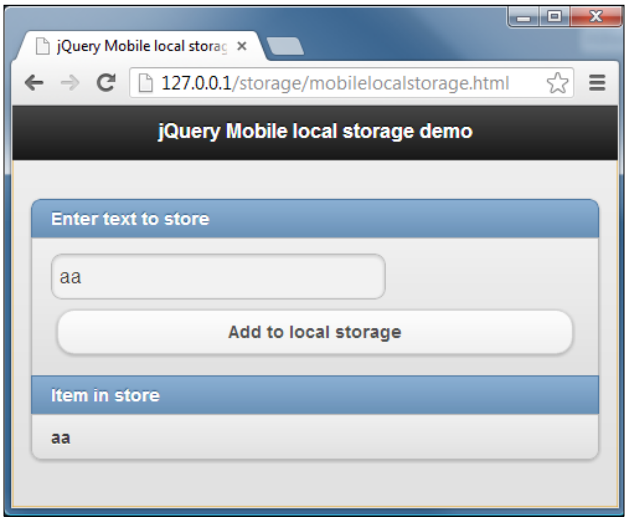
```
<style>
  #message { display: none; border-radius: 10px;
  color: #fff; padding: 10px; background-color: #ff0000; }
  #storageDemoPage h2 { white-space: normal; }
</style>
```

4. Finally we need to tie the code together and provide the mechanism to add it to `LocalStorage`. Add the following code snippet into the `<head>` area of your code:

```
<script type="text/javascript">
  $(document).ready(function() {
    var localStorageKey = "demoStorage";
```

```
    if (Modernizr.localstorage) {
      displayValue();
    } else {
      $('#message').text("Sorry, but your browser doesn't
      support localstorage!").show();
      $('#addToStorage').attr('disabled', 'disabled');
    }
    $('#addToStorage').click(function(e) {
      localStorage.setItem(localStorageKey,
      $('#entry').val());
      displayValue();
      e.preventDefault();
    });
    function displayValue() {
      var item = localStorage.getItem(localStorageKey);
      if (item == null || item == 0) {
        item = 'Nothing in store,
        or store has an empty value';
      }
      $('.storeItem').text(item);
    }
  });
</script>
```

5. If all is well, you should get the result as shown in the following screenshot, when previewing in your browser:



How it works...

A closer look at the code should show that there are no differences between a normal platform and a mobile one, when handling Local Storage. In this instance, we've used Modernizr and jQuery Mobile to check the compatibility and style the form for a mobile platform. We then added function calls to manage Local Storage; these functions are not different from those in the earlier examples, although here we have used jQuery to handle them, not pure JavaScript. The first method handles storing the content into Local Storage and then automatically updates the page to show the stored contents. The second function handles any instance where the storage or contents of the input field are empty.

Now that we have seen how we can use jQuery with Local Storage, let's focus on creating a simple plugin that you can use in your projects, as part of the next recipe.

Building a simple plugin using jQuery (Should know)

Up until now, we've worked with Local Storage directly in our code. While this will work perfectly well, it's not an ideal solution; it improves messy code if it is held inline, and adds a lot of unnecessary code to a site. A better option is to separate the code and turn it into a plugin that we can re-use; we will see how we can build a simple plugin as part of this recipe.

Getting ready

For this recipe, all you will need is your text editor. Anything else that is required is called directly from within the code.

How to do it...

Perform the following steps, to build a simple plugin using jQuery:

1. Let's begin with adding the following code snippet to a new HTML document, and save it as `basicplugin.html`:

```
<!DOCTYPE html>
<head>
  <title>Basic plugin test</title>
  <meta charset="utf-8">
  <script src=
  "http://code.jquery.com/jquery-1.8.3.min.js"></script>
  <script src="jquery.alStorage.js"></script>
</head>
<body>
  <div id="demoresults">
```

```
    <div id="details">
      <h2>Demo using alStorage</h2>
      <p></p>
    </div>
  </div>
</body>
</html>
```

2. We need to provide the script that will save some content to `LocalStorage`, and then retrieve it, so add the following code in between the `<p>` tags:

```
<script type "text/javascript">
  var testItem = "Hello, my name is Alex.";
  $.alStorage.setItem("testObject", testItem);
  document.write("The value in the 'testObject' key has
  been saved to LocalStorage.<p>")
  var retrievedObject =
  $.alStorage.getItem("testObject");
  document.write("The retrieved value is: " +
  retrievedObject);
</script>
```

3. We still need to perform a couple of steps for getting a working example. As it stands, it won't win any awards for styles, so let's add some basic styles to at least make it look presentable! Add the following code into the `<header>` area of our document. We've included a Google web font for good measure:

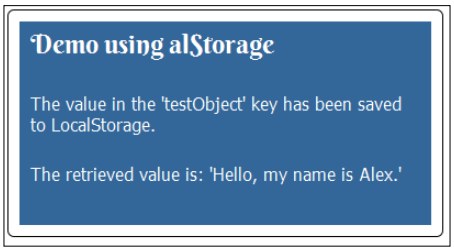
```
<link rel="stylesheet" type="text/css" href="http://fonts.
googleapis.com/css?family=Berkshire+Swash" >
<style type="text/css">
  body { color: #FFF; }
  h2 { padding: 5px; margin-top: -5px;
  font-family: 'Berkshire Swash', cursive; }
  p { padding: 5px; font-family: tahoma,
  arial, sans-serif; }
  #demoresults { border: 1px solid black; border-radius:
  6px; height: 240px; padding: 0 10px 5px; width: 375px; }
  #details { background-color: #369; height: 89.5%;
  margin-top: 10px; padding: 5px; }
</style>
```

4. Here's where the meat and bones of this demo lie. Add the following code to a new text document, and save it as `jquery.alstorage.js`; this provides the iQuery functions that are called from the main code:

```
(function($){
  $.alStorage = {
    setItem:
```

```
        function(key,value){localStorage.setItem(key,value)},
        getItem:
        function(key){return localStorage.getItem(key)},
        removeItem:
        function(key){localStorage.removeItem(key)},
        clearItem: function(key) {localStorage.clear()}
      }
})(jQuery);
```

5. If all is well, you should see something similar to the following screenshot when previewing this in your browser:



How it works...

In this recipe, we've taken the normal methods available for Local Storage and moved them into a jQuery plugin. We've encased each method call into a method handler, so that from within our code, we can call the appropriate method in the same way, as if the methods had been within the main code and not in a separate plugin file. To demonstrate it, we've used the `setItem` method for the phrase `Hello, my name is Alex` with a `testObject` key in Local Storage. We've then retrieved the text from the same value using the `getItem` method.

There's more...

This little plugin serves as a perfect illustration of how you can encase Local Storage into a jQuery plugin. You're probably thinking, "Great, I can use this in my code; I'm good to go now." Except probably not.

Huh? It might seem odd to say that, but there is a good reason—there are some points to consider here:

- Although this plugin works, it does not follow the best practice. It was meant to illustrate how you could encapsulate Local Storage within a jQuery plugin. You would very likely want to redesign it to follow accepted best practice. For a good starting point, you may want to take a look at *JQuery 1.4 Plugin Development Beginner's Guide by Giulio Bai, Packt Publishing*.

- There is no scope within this plugin to cover Session Storage as it stands. You can easily rewrite this plugin to include support for Session Storage too. This is important, as you will need to store content in the browser that would only need to be there for the current session—using Local Storage isn't appropriate in this instance.
- A benefit of using a plugin is that you can provide consistent support for Local Storage, while abstracting the code required for using it natively, or falling back to a polyfill for older browsers. The important thing to bear in mind here is, do you need to provide fallback support? This all depends on the browsers that your visitors are using to view your site. If they are all using modern browsers, there is little need for providing an extra layer of code, when the native equivalent will suffice. If you do still have older browsers being used (which is more likely), a plugin will serve you well.
- Although you can write your own plugin, it may be more sensible to look and see what is available for use already; several people have created plugins and made them available for download. It may be a better use of your development time to see what is available and whether it suits your needs or not, before going ahead with developing your own plugin.

That last consideration raises an important point. It may be a better use of your time to complete some research with a view to potentially using a plugin that someone has already made available for use, and which already contains additional functionality that you are looking to use in your project. A good example of this could be to provide TTL support for Local Storage, as well as support for storing arrays in Local Storage; as it so happens, there is a plugin that does both of these and more. In the next recipe, we're going to take a look at jStorage, and why you are not always forced to use strings when storing content!

Adding objects, arrays, and TTL support to storage (Become an expert)

At the very start of this book, I mentioned the need to use strings to store content in Local Storage. What if I said that it was possible to store objects and arrays in Local Storage as well? You'd be right in pointing out that these are not strings. You can still store content from data objects and arrays, if you use the power of JSON to convert them, as we will see in this recipe.

Getting ready

For this recipe, we will need our usual text editor, as well as the code that accompanies this book. We will also need a copy of the jStorage plugin that is available from `http://github.com/andris9/jStorage/raw/master/jstorage.js`, as well as jQuery; you can download this from `http://code.jquery.com/jquery-1.8.3.min.js` (or use the link as a CDN link).

How to do it...

Perform the following steps, for adding objects, arrays, and TTL support to storage:

1. Let's begin by adding the following code to a new HTML document. Then, save it as `addttlsupport.html`:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Getting Started with HTML5 local storage</title>
    <meta charset="utf-8">
    <script src=
    "http://code.jquery.com/jquery-1.8.3.min.js"></script>
    <script src="jstorage.js"></script>
    <script type="text/javascript" src="addttlsupport.js">
    </script>
    <link rel="stylesheet"
    type="text/css" href="addttlsupport.css">
  </head>
  <body onload="load()">
  <div id="myform">
    <div id="results">
      <b>Simple form with name and age:</b><p>
      <div id="gTTL"></div>
    </div>
  </div>
  </body>
</html>
```

2. We also need to include the CSS styling for the page. Download the `addttlsupport.css` file from the code that accompanies this book, and save it in the same location as the main HTML file.

3. We now need to add the basic framework for inputting values and calling the appropriate method handlers:

```
  <div id="gTTL"></div>
    <table>
      <tr>
        <td>Name:</td>
        <td><input type="text" id="name"></td>
      </tr>
      <tr>
        <td>Age:</td>
        <td><input type="text" id="age">
      </tr>
    </table>
```

```
<p>
  <table>
    <tr>
      <td><input type="button" class="button medium
      gray" id="addInfo" value="Add to Storage"></td>
      <td><input type="button" class="button medium
      gray" id="getInfo" value="Get from Storage"></td>
      <td><input type="button" class="button medium
      gray" id="gTTLbutton" value="Get TTL"></td>
    </tr>
    <tr>
      <td><input type="button" class="button medium
      gray" id="saveC" value="Save complex data"></td>
      <td><input type="button" class="button medium
      gray" id="restoreC" value="Restore complex
      data"></td>
   </tr>
</table>
```

4. We need to add the code to handle the various functions required to manage `LocalStorage`. Go ahead and add the following into a file called `addttlsupport.js`, beginning with the `save()` method handler:

```
function saveInfo() {
  try {
    $.jStorage.set("name", $("#name").val());
    $.jStorage.set("age", $("#age").val());
    $("#name").val("");
    $("#age").val("");
    $.jStorage.setTTL("name", 30000);
  }
  catch (e) {
    if (e == QUOTA_EXCEEDED_ERR) {
      console.log("Error: Local Storage limit exceeds.");
    } else {
      console.log("Error: Saving to local storage.");
    }
  }
}
```

5. Next comes the method handler to retrieve content from `LocalStorage`:

```
function getInfo() {
  console.log("Getting your data from local storage.");
  $("#name").val($.jStorage.get("name"));
  $("#age").val($.jStorage.get("age"));
}
```

6. We must not forget the check to ensure that we can even handle Local Storage. At this point, you should open the console in your browser, to see the output:

```
function load() {
  if (typeof(Storage) == "undefined" ) {
    alert("Your browser does not support HTML5
    localStorage. Try upgrading.");
  } else {
    console.log("Both localStorage and sessionStorage
    support is there.");
  }
}
```

7. This method handler is responsible for saving data objects to `LocalStorage`.

```
function sComplexData() {
  console.log("Saving complex data to local storage.");
  var personObject = new Object();
  personObject.name = $("#name").val();
  personObject.age = $("#age").val();
  $.jStorage.set("person", JSON.stringify(personObject));
}
```

8. And this one for retrieving them:

```
function rComplexData() {
  console.log("Restoring complex data from local
  storage.");
  var personObject =
  JSON.parse($.jStorage.get("person"));
  $("#name").val(personObject.name);
  $("#age").val(personObject.age);
}
```

9. Almost last but by no means least is the method for getting the TTL value for a named key:

```
function getTTLforName() {
  var ttl = $.jStorage.getTTL("name");
  $("#gTTL").html("TTL time for 'name' key,
  using simple store: " + ttl);
}
```

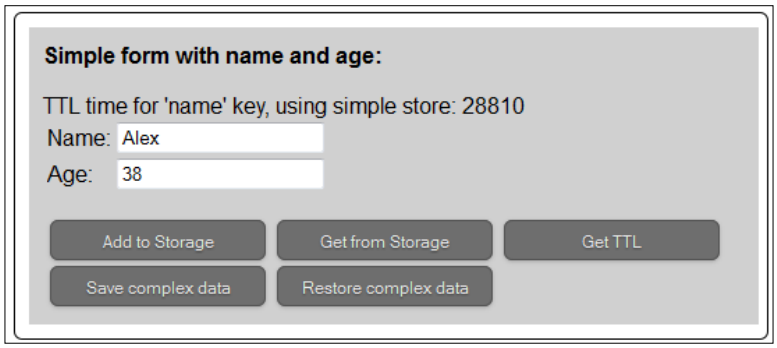
10. We finally need to add one more script. This provides all of the calls for each button:

```
  <script type="text/javascript" src="addttlsupport.js">
  </script>
  <script type="text/javascript">
    $("document").ready(function() {
```

```
    $("#getInfo").click(getInfo);
    $("#addInfo").click(saveInfo);
    $("#gTTLbutton").click(getTTLforName);

    $("#saveC").click(sComplexData);
    $("#restoreC").click(rComplexData);
  })
</script>
```

11. If all is well, you will see the following result when previewing this in your browser. This has already had the values `Alex` and `38` added to `LocalStorage`, using Add to Storage and the TTL value retrieved:



How it works...

Read through the code carefully, do you notice anything in particular? One thing may pop out. We've used JSON to convert contents into strings before storing them. The keen-eyed amongst you would notice that we have actually used the same principles within the very first recipe in this book, *Basic use of Local Storage*.

We begin with the usual call to jQuery, followed by one to jStorage. We then have a number of functions that are used to either save content to, or retrieve from, `LocalStorage`. Our first two methods, `save()` and `get()`, are responsible for fetching and retrieving content from `LocalStorage`; the former includes a call to `.setTTL()` to set a TTL value `30000` or 30 seconds.

The next one, `load()`, performs a check when loading the web page, to ensure we use `LocalStorage` correctly. This is followed by `sComplexData()`, which creates a data object called `personObject()`, stores two values person and age within it, and then converts them into a key called `person`, using JSON, before storing them in `LocalStorage`. The `rComplexData()` handler retrieves the same `person` key, then parses it using JSON, before extracting the appropriate values and displaying them on the screen.

The keen-eyed amongst you may have noticed that the code does not include support arrays. This is deliberate; I thought I would leave this as a recipe for you to complete! A hint, it uses the same method to store the content in `LocalStorage`; all you need to do is get the content into an array first.

So far, we've looked at using strings within our code, which works well. What if I say we could really turn the tables, and start to include images within `LocalStorage`? Yes, you heard right, images! We'll see how we can do that as part of the next recipe.

Storing images within Local Storage (Become an expert)

Up until now, we've worked with storing text values in `LocalStorage`. I hope you agree that it is an easy process to perform, and that used with care, we can achieve some great results. I mentioned though, at the end of the last recipe, that we could really turn the tables, and start to store images in `LocalStorage`. We'll see how to achieve this, using a technique developed by Rob Nyman, a Technical Evangelist at Mozilla.

Getting ready

For this recipe, we will need a suitable image to store in `LocalStorage`. There is no hard and fast rule on size, but we should bear in mind that in reality `LocalStorage` can't hold more than 5 to 10 MB (depending on the browser), so it is wise to choose a sensible size. I will assume you have selected a suitable image, which is ideally `250px` in width and `377px` in height, so as to avoid the need to alter the CSS styling used in this recipe.

You will also need a copy of the code that is available with this book, and of course your trusty text editor, as always!

How to do it...

Perform the following steps, for storing images within `LocalStorage`:

1. Create a new folder called `image` in `localstorage`, and save this on your hard disk. Inside this folder, create two subfolders called `js` and `css`.
2. From the code that accompanies this book, you will need to extract the `lsimage.html` file from `image` in the `localstorage` folder, and save it in the folder we've just created.

3. In your text editor, go ahead and create the following JavaScript file, saving it as `base.js`, in the `js` subfolder. We'll go through it block by block, starting with declaring a number of variables:

```
(function () {
// localStorage with image
var storageFiles =
JSON.parse(localStorage.getItem("storageFiles")) || {},
elephant = document.getElementById("elephant"),
storageFilesDate = storageFiles.date,
date = new Date(),
todaysDate = (date.getMonth() + 1).toString() +
date.getDate().toString();
```

4. We need to check if the `LocalStorage` area exists already; if not, go ahead and create a new one, set up the canvas element, and draw the image on it:

```
if (typeof storageFilesDate === "undefined" ||
storageFilesDate < todaysDate) {
  elephant.addEventListener("load", function () {
    var imgCanvas = document.createElement("canvas");
    var imgContext = imgCanvas.getContext("2d");

    imgCanvas.width = elephant.width;
    imgCanvas.height = elephant.height;
    imgContext.drawImage(elephant, 0, 0, elephant.width,
    elephant.height);
```

5. Now that the image has been rendered on the screen, let's save it as a data URL, add the date of creation, and convert it into a string before uploading it to `LocalStorage`:

```
    storageFiles.elephant =
    imgCanvas.toDataURL("image/png");
    storageFiles.date = todaysDate;
    try {
      localStorage.setItem("storageFiles",
      JSON.stringify(storageFiles));
    }
    catch (e) {
      console.log("Storage failed: " + e);
    }
  }, false);
    elephant.setAttribute("src", "elephant.png");
  }
```

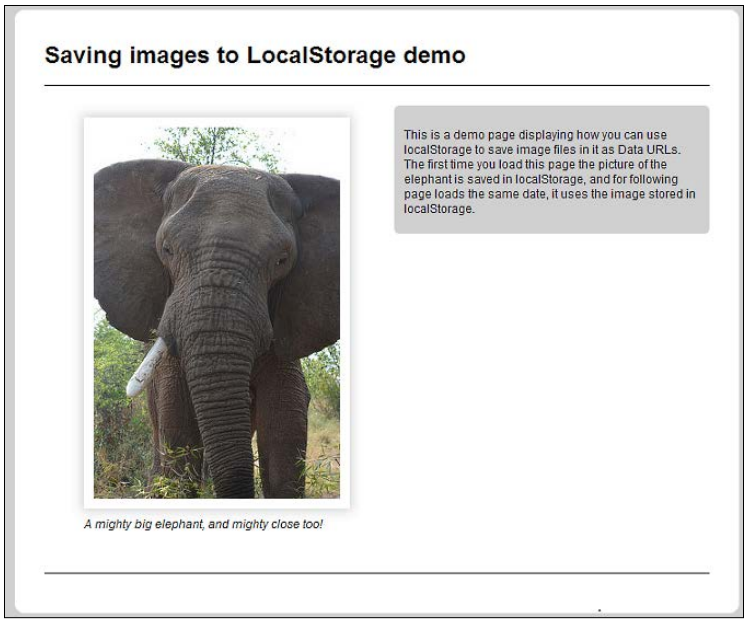
6. However, if the image already exists, we use the one from `LocalStorage`:

```
    else {
      elephant.setAttribute("src", storageFiles.elephant);
    }
  })();
```

7. Finally we need to add some styling, so the page appears presentable. Save the following code to a new file called `base.css`, stored in the `css` subfolder:

```
body { font: 12px "Helvetica Neue", "Helvetica", sans-serif;
background: #ccc; }
h1 { margin-top: 0; }
img { padding: 10px; box-shadow: 0 0 10px 2px #ccc; margin-bottom:
5px; }
figcaption { font-style: italic; }
header { margin-bottom: 20px; border-bottom: 1px solid #000; }
.container { width: 675px; background: #fff; border-radius: 10px;
margin: 0 auto; padding: 30px; }
.main-content { overflow: hidden; }
.additional-content { float: right; width: 300px; height: 110px;
padding: 10px; background: #ccc; border-radius: 5px }
.page-footer { border-top: 1px solid #000; margin-top: 30px;
padding-top: 10px; }
```

8. If all is well, you will see something similar to the following result, when previewing this in your browser:



How it works...

In this recipe, the code of particular interest is in the `base.js` file we've created for this demo. We start by checking to if our `LocalStorage` element is already present. If so, the code uses the image directly from `LocalStorage`, otherwise it sets up a number of variables for storing values such as the date on which the image was stored, today's date, and of course the image is converted to a string using JSON.

We then do a check to confirm dates and create `LocalStorage` based on the outcome. If all is well, we can go ahead with first creating a canvas element and then drawing the full image on the canvas. We save a copy of the image as a data URL, and set the date. It is then converted to a string using JSON and finally stored (provided it is not too big).

There's more...

This is a very powerful method, but that should be used with care. Local Storage was not designed to handle huge images! This said, it will be very useful, particularly for images that rarely change, such as navigation buttons or company logos. A good method to use would be converting any images into sprites. This could help reduce the number of different requests required, and potentially the size of the file being held.

A number of developers have taken this principle a step further, and created cache plugins for storing images in `LocalStorage`. Visit `http://www.jquery4u.com/plugins/10-jquery-image-cache-plugins-scripts/` if you would like to explore this functionality further.

Okay, before moving onto looking at our next demonstration, there is one area we must take a look at, and upon which I am sure you will ask questions. It relates to the amount of storage area available within `LocalStorage`, as we will see in the next recipe.

Adjusting the Local Storage space for browsers (Must know)

As you develop more with Local Storage, one question that is likely to come to your mind is, "Can I increase the space I can use with Local Storage?" If you're hoping for yes as an answer, unfortunately you may be sorely disappointed. The limits are fairly strict, although there is an exception, as you will see in this task.

Getting ready

For the purposes of this test, you will need to launch a session in any of the browsers that you wish to test, such as Opera, Internet Explorer, Safari, Firefox, and Chrome. I will assume we are using Firefox to run through the initial test.

How to do it...

For adjusting the Local Storage space for browsers, perform the following steps:

1. Let's make a start by opening a session in Firefox. Then browse to `http://dev-test.nemikor.com/web-storage/support-test/` and click on **Run Web Storage Tests**.
2. Run the tests through to completion. When you see **Your results have been saved** in a dialog box, the tests are completed.
3. The tests will show you the maximum amount of characters you can store with the space available on the basis of using UTF-16 format for JavaScript strings, assuming that you are using a modern browser that supports Local Storage.

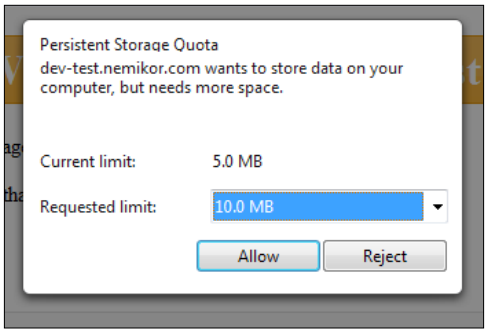
How it works...

In this test, we've asked the site to see if your chosen browser first supports Local, Global, or Session Storage. If it does, it will then see how many characters it can safely store in each area, before a DOM error of type `QUOTA_EXCEEDED_ERR` would be thrown. If a particular type of storage capability is not supported, this test is not run. (In most cases, Global Storage was really only supported by Firefox, up to and including version 13.)

There's more...

As part of researching this recipe, I ran some tests to get an idea of what each of my browsers supported. I used Firefox 17, Chrome 23, IE 9, Safari 5, and Opera 12.

Most of the browsers were consistent, in as much as they are limited to a set size for Local Storage, although the exact limit varies between browsers. For example, IE 9 and Firefox 17 will hold up to 10 MB, while Safari and most mobile platforms are still limited to a maximum of 5 MB. The only exception to this was Opera. Using the test from this recipe, I was able to get it to hold up to 40 MB! It does correctly prompt you though for permissions:



All of the four browsers supported an unlimited amount of space for Session Storage, although this will disappear when the browser session is refreshed.

Ok, let's move on now, and turn our attention to focusing on some real-life examples where you can use Local Storage, beginning with constructing a stickies option for using in a browser.

Building a stickies option for using in a browser (Become an expert)

We begin the first of our demos. This looks at building a basic stickies option, as a practical introduction to using Local Storage.

Getting ready

For this recipe, you will need your normal text editor, as well as the code that accompanies this book.

How to do it...

Perform the following steps, to build a stickies option for using in a browser:

1. Let's begin with creating a new HTML document. Save the following code as `stickies.html`:

```
<!DOCTYPE html>
<head>
  <title>Storing content using Stickies and
  LocalStorage</title>
  <meta charset="utf-8" />
  <link href=
```

```
    "http://fonts.googleapis.com/css?family=
    Reenie+Beanie:regular" rel="stylesheet" type="text/css">
    <link rel="stylesheet" type=
    "text/css" href="css/stickies.css">
    <script type="text/javascript" src=
    "http://code.jquery.com/jquery-1.8.3.min.js"></script>
    <script type="text/javascript"
    src="js/stickies.js"></script>
  </head>
  <body>
    <div id="postit-pad">
      <pre id="postit" contenteditable="true"
      onkeyup="storeSticky(this.id);"></pre>
      <a class="clearStorage" href='' onclick=
      'javascript:clearLocal();'>Clear local storage</a>
    </div>
    <script>
      getSticky();
    </script>
  </body>
  </html>
```

2. We need to add the functionality that's going to save content to, or retrieve from, `LocalStorage`. Go ahead and add the following into a new document called `stickies.js`, in the subfolder `js`:

```
function storeSticky(id) {
  var scribble = $("#postit").text();
  localStorage.setItem('userScribble',scribble);
}
function getSticky() {
  if ( localStorage.getItem('userScribble')) {
    var scribble = localStorage.getItem('userScribble');
  }
  else {
    var scribble = 'You can scribble directly on this
    sticky...and I will also remember your message the next
    time you visit!';
  }
  document.getElementById('postit').innerHTML = scribble;
}
function clearStorage() {
  clear: localStorage.clear();
  return false;
}
```

3. We finally need to add some styling. In the code that accompanies the book, you will find two additional folders for this project that we need. Copy `img` and `css` into the same location as your HTML and JavaScript files.
4. If all is well, you should see something similar to the screenshot, when previewing this in your browser:

How it works...

This is a nice simple example of how you can use Local Storage to create a useful application, even if ours would need more work before it becomes a polished saleable application! In this instance, we've used a mix of plain JavaScript and jQuery—not out of necessity, but as a means to illustrate that you can achieve the same result using either method.

We've created a simple framework within a `<div>` element called `postit-pad`, which is styled using CSS2 and CSS3 rules. Within the `stickies.js` file, we've created three simple functions—one to save content to `LocalStorage`, one to retrieve it, and one to remove it from `LocalStorage` (if desired). All three still use the methods that we've covered in this book. The difference here is that we are using native methods, rather than a polyfill or external library.

There is scope to improve this code. For example, it is missing vital checks to ensure compatibility. If you would like to look at what is possible, you may want to take a look at the following two posts:

- `http://acuriousanimal.com/blog/2011/08/12/local-storage-storing-sticky-notes-on-your-machine-with-html5/`
- `http://net.tutsplus.com/tutorials/html-css-techniques/building-persistant-sticky-notes-with-local-storage/`

Let's move on and take a look at another demo, which is more of a proof of concept, and will show you how you can use Local Storage to create a basic to-do list, as you will see in the next recipe.

Building a simple to-do list (Become an expert)

While researching material for this book, I came across a great post by Koes Bong at `http://web.koesbong.com/2011/01/24/sortable-and-editable-to-do-list-using-html5s-localstorage/` on creating a sortable to-do list using Local Storage. It shows off a number of aspects of using Local Storage that we have covered in this book perfectly. Let's take a look at how.

Getting ready

For this task, you'll need a few things in addition to your normal text editor and the code that accompanies this book. I've assumed that local copies of all of the following plugins have been saved for the purposes of this task:

- **jQuery pub/sub plugin**: You can download this from `https://github.com/phiggins42/bloody-jquery-plugins/blob/master/pubsub.js`. You will need to click on the **Raw** button, and then save the page that appears as a JavaScript file.
- **jQuery inline edit plugin**: This is available from `https://github.com/caphun/jquery.inlineedit/archive/master.zip`.
- **A cross button image**: Any will do—if possible, try to aim to get one that is about `25px` square, in PNG format. If yours is different, you will need to edit the code accordingly. For this recipe, I've assumed you have something in PNG format.
- **Background image (used on each list item)**: This is available as part of the jQuery UI download, which you can get from `http://jqueryui.com/resources/download/jquery-ui-1.9.2.custom.zip`. Look within `\jquery-ui-1.9.2.custom\css\smoothness\images` for the image.

All credit for the original code should go to Koes Bong. I've merely updated it to use more recent versions of jQuery and some of the plugins, as well as updating some of the styles, to give a more polished look.

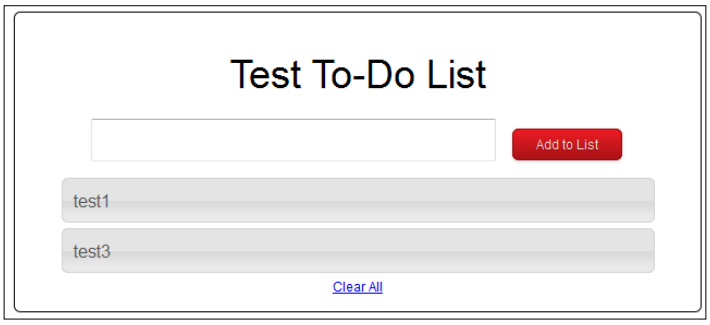
How to do it...

Perform the following steps, for building a simple to-do list:

1. Let's get all of the files in place first, so create a folder called `To-Do List` somewhere on your PC, and save copies of the following folders from the code available with this book: `css`, `img`, and `js`.
2. In your text editor, add the following code to a new HTML document and save it as `todolist.html` within the `To-Do List` folder:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Test To-do List using Local Storage</title>
    <meta charset="utf-8" />
    <link rel="stylesheet" href="css/base.css"
    type="text/css" />
  </head>
  <body>
    <div id="container">
      <h1>Test To-Do List</h1>
      <form id="todo-form">
        <input id="todo" type="text" />
        <input id="submit" class="button medium red"
        type="submit" value="Add to List">
      </form>
      <ul id="show-items"></ul>
      <a href="#" id="clear-all">Clear All</a>
    </div>
    <script src="js/jquery-1.8.3.min.js"></script>
    <script src="js/jquery-ui.js"></script>
    <script src="js/jquery.inlineedit.js"></script>
    <script src="js/pubsub.js"></script>
    <script src="js/base.js"></script>
  </body>
</html>
```

3. If all is well, you should see something similar to the following, when previewing it in a browser. Here, I've already added two items to the list:



How it works...

Although there may appear to be a fair chunk of code in this recipe, most of the magic is in the `base.js` file. This includes all of the method handlers to manage the storing and retrieving of content from Local Storage.

If we look at the script closely, we see that it starts with a request to retrieve content from `LocalStorage`, which is then fed into a `for` loop to populate the list shown on screen. The methods that follow this are then responsible for adding, removing, or sorting content accordingly. The code then contains some additional functionality to allow in-line editing of each task using the `jquery.inlineedit.js` plugin, as well as functionality to fade in and out the X logo on each task item. Finally the code finishes with a method to manage the saving of content to `LocalStorage`, using the jQuery pub/sub plugin from Peter Higgins.

There's more...

This is very much a proof of concept. You may well find it doesn't work in Internet Explorer, for example! It is worth studying the code though—there are some great examples of how you can use jQuery to manage Local Storage. I would suggest though that you will want to work on it if you use it in your own projects—it needs altering to make compatible with IE; it could also be turned into a plugin, which will certainly make it more useful.

So far, we've looked at how you can use Local Storage in a web application of some description. What about using it in a content management system or blog? This is equally possible; over the next two recipes, we'll take a look at how you can achieve both, beginning with implementing it in a simple "Contact Us" form.

Using Local Storage in a form (Become an expert)

How many times have you filled out a form, only to find you lost an Internet connection, and have to restart afresh? Irritating, isn't it? No problem, we can easily fix this, using Local Storage. In this recipe, we're going to take a look at how you can use it within a simple demo form.

Getting ready

For this recipe, all we'll need in addition to our text editor is one image. It's to provide some background color, and a copy of it is present in the code that accompanies this book. I would strongly recommend creating a new folder to store your demo within. I will assume you have done this for the purposes of this demo.

How to do it...

Perform the following steps, for using Local Storage in a form:

1. OK, let's make a start by saving the following code to a new HTML document, as `useinforms.html`:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>HTML5 localStorage Demo</title>
  </head>
  <body>
  </body>
</html>
```

2. We need to use jQuery, some CSS, and a custom script, so we need to include references to all three. We will create the appropriate files later in this recipe:

```
<link href="css/useinforms.css"rel="stylesheet" type= "text/css" >
<script src=http://code.jquery.com/jquery-1.8.3.min.js type="text/
javascript"></script>
<script type="text/javascript" src="js/useinforms.js" type="text/
javascript"></script>
```

3. Next we need to add our simple form, so add the following code between the `<body>` tags:

```
<form id="localStorageTest" method="post" action="">
  <label>Name:</label>
  <input type="text" name="name" id="name"
```

```
    class="stored" value="" />

    <label>Email:</label>
    <input type="email" name="email" id="email"
    class="stored" value="" />

    <label>Message:</label>
    <textarea name="message" id="message"
    class="stored"></textarea>

    <input type="submit" class="demo-button" value="Submit" />
</form>
```

4. We finally need to add the script that will add the values from each form field into `LocalStorage`. Go ahead and save the following code to a new file called `useinforms.js`, within a subfolder called `js`:

```
$(document).ready(function () {
  function init() {
    if (localStorage["name"]) {
      $('#name').val(localStorage["name"]);
    }
    if (localStorage["email"]) {
      $('#email').val(localStorage["email"]);
    }
    if (localStorage["message"]) {
      $('#message').val(localStorage["message"]);
    }
  }
  init();
});
$('.stored').keyup(function () {
  localStorage[$(this).attr('name')] = $(this).val();
});
$('#localStorageTest').submit(function() {
  localStorage.clear();
});
```

5. It's going to look pretty ugly without some form of styling, so let's go ahead and fix that by adding the following code to a new file called `useinforms.css`, within a subfolder called `css`. We begin with resetting some of the base styles:

```
html, body, div { border:0; font:inherit; font-size:100%;
margin:0; padding:0; vertical-align:baseline; }
html, body { height: 100%; }
```

6. Next comes the background. Set styles for paragraphs and the form. The `.clear` rule is used to display the fields in alignment:

```
body { background: url(../images/noise.png) rgb(241, 241, 241);
color: #40382B; font-family: 'Tahoma', sans-serif; font-size:
100%; margin-left: 20px; margin-top: 10px; }
p {font-size: 1em; margin-bottom: 1.25em; line-height: 1.4em;
text-align: left;}
form { padding: 10px; width: 500px; border: 1px solid black;
border-radius: 4px; }

.clear {clear: both;}
```

7. We also need to provide some styles for the labels and input fields used:

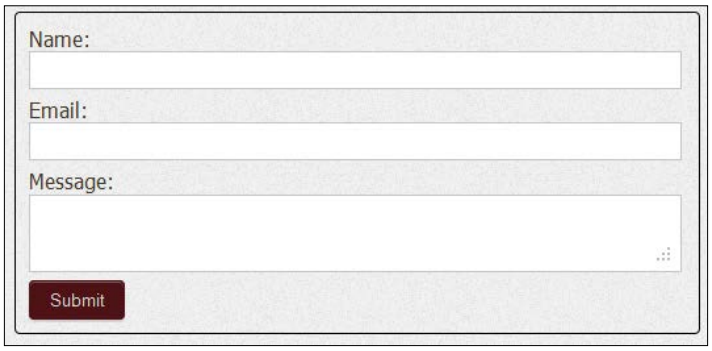
```
#localStorageTest label { display: block; }
#localStorageTest input[type=text],
#localStorageTest input[type=email] { display: block; margin-
bottom: 10px; width: 97%; border: 1px solid #C5C5C5; margin-
bottom: 6px; padding: 5px !important; font-size:     0.9em; }
#localStorageTest textarea { border: 1px solid #C5C5C5; display:
block; font-size: 0.9em; margin-bottom: 6px; padding: 5px
!important; width: 97%; max-width: 97%;}
```

8. Finally, we want to make the submit button look a little more attractive:

```
.demo-button { padding: 5px 12px; background: #4e1216; color:
#ddd; -webkit-border-radius: 4px; -moz-border-radius: 4px;
border-radius: 4px; border: 1px solid #4B1115; text- decoration:
none; cursor: pointer;
-webkit-box-shadow: inset 0 1px 0 rgba(255, 255, 255, 0.4), 0 1px
1px rgba(0, 0, 0, 0.2); box-shadow: inset 0 1px 0 rgba(255, 255,
255, 0.3), 0 1px 1px rgba(0, 0, 0, 0.2); }

.demo-button:hover { color: #fff; background: #7F5949; border:
1px solid #4B1115; text-decoration: none; }
.demo-button:active { -webkit-box-shadow: inset 0 1px 4px rgba(0,
0, 0, 0.6); -moz-box-shadow: inset 0 1px 4px rgba(0, 0, 0, 0.6);
box-shadow: inset 0 1px 4px rgba(0, 0, 0, 0.6); background:
#4e1216; border: 1px solid #4B1115; }
```

9. Alright, if all has gone without any issues, you should see something similar to the following result, when previewing this in a browser:



How it works...

Although there are a few lines of code involved in this recipe, the concept behind them is very simple. We begin by creating our three-field form to act as our framework.

The real magic happens in the accompanying JavaScript file. We create a function called `init`, which retrieves the values for name, e-mail, and message from the `LocalStorage` area of the browser. These details will then be shown when you open your browser. Any subsequent change to the contents of each field is automatically saved into the `LocalStorage` area, using the `.keyup()` event handler. Finally, when we submit the form, we clear `LocalStorage`. After all, once you've submitted your form, it's unlikely you will still want details being shown!

We've had a look at a simple form in this recipe, in which we used Local Storage to help prevent loss of information, should the unthinkable happen and you lose your connection to the Internet. That's good, but I know some of you will likely be users of CMS-based systems, and will want to know if the same principle works there too. The good news is that it does. We'll take a look at how that is done, using WordPress as an example, in the next recipe.

Using Local Storage in a CMS (Become an expert)

So far, we've looked at using Local Storage in our websites. This assumes that they have been built from scratch over a period of time. However, we cannot afford to ignore CMS systems, particularly popular ones such as WordPress. The great thing is that a plugin for WordPress has been developed to take advantage of Local Storage. We're going to take a look at this plugin as part of this recipe.

Getting ready

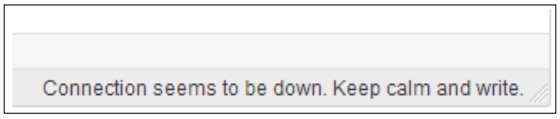
For this task, you will need access to a working installation of WordPress, either remotely or using a local web server such as WAMP Server. I will assume you are using an instance of the latter for the purposes of this recipe (currently at version 3.5 at the time of writing this book). You will also need a copy of the open source Local Storage Backup plugin, which is at version 0.92 (at the time of writing this book), and is available to download from `http://wordpress.org/extend/plugins/local-storage-back-up/`.

I would strongly recommend using Firefox for the purposes of the test at the end of this recipe. It handles the offline connection more gracefully than IE!

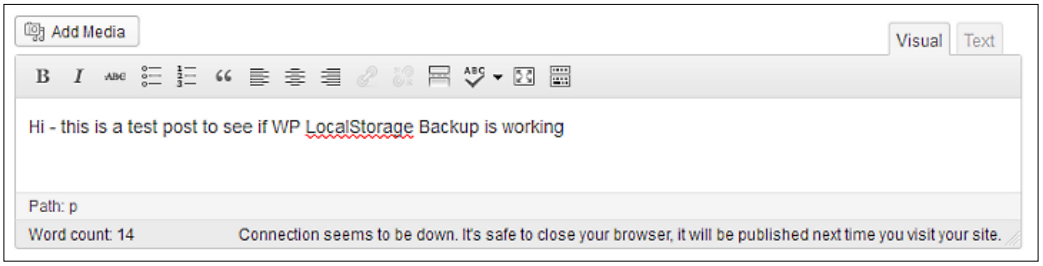
How to do it...

Perform the following steps, for using Local Storage in a CMS:

1. Browse to your Admin area in WordPress, and then install the plugin in the normal way. If you are unfamiliar with this process, visit the WordPress Codex at `http://codex.wordpress.org/Managing_Plugins`. There you will find details on how to install plugins.
2. To test that it is running, browse to **Posts**, then **Add New**, and write a post as normal. Make sure you include a title as well! Before publishing it, set the browser into offline mode and click on **Publish**. Instead of showing a 404 error, WordPress will show the following result, in the bottom-right corner of the post window:



3. This confirms the plugin is working. The message will then change as shown in the following screenshot:



4. Switch your browser back to online mode and click on **All Posts** to force a refresh. Your browser will show a message, confirming that posts still need to be published:

How it works...

This is one of those plugins for WordPress that does a simple job well. It sets WordPress to divert any requests for publishing posts to `LocalStorage`, if it determines that you are working offline as a result of a dropped connection. As soon as your Internet access has resumed, it will recognize that there are posts that need to be published, and prompt you to complete this process.

There's more...

It should be noted that this plugin isn't designed to handle comments in a similar fashion. If you want to include this facility, take a look at another plugin called WP Local Storage, which is available at `http://wordpress.org/extend/plugins/wp-local-storage/`.

We're almost at the end of our journey through Local Storage. Before we finish, there is one more area we are going to look at, which relates to forms. Ask yourselves this question, "How many times have you visited a site as a registered user, only to find that it still thinks you are a new user?" Sounds familiar? We'll see how to avoid this, as part of the next recipe.

Using Local Storage to hide sign-up forms (Become an expert)

If you browse on the Internet and register for access to a site, you would be more than a little irritated if the same site kept prompting you to register, wouldn't you? This is one of my pet hates. We can easily fix this with the help of Local Storage, as you will see in this recipe.

How to do it...

Perform the following steps, for using Local Storage to hide sign-up forms:

1. Let's begin by creating our basic framework. Add the following to a new HTML document, and save it as `hideform.html`:

```
<!DOCTYPE html>
<html>
  <head>
```

```
    <meta charset="utf-8">
    <title>Hiding a sign-up form demo</title>
  </head>
  <body>
  </body>
</html>
```

2. We need to add references to Modernizr, jQuery, and our custom script. We're going to add a Google font to style the form, for good measure:

```
<link href="hideform.css" rel="stylesheet">
<link href="http://fonts.googleapis.com/
css?family=Shadows+Into+Light" rel="stylesheet" type="text/css">
<script type="text/javascript" src="http://code.jquery.com/jquery-
1.8.3.min.js"></script>
<script type="text/javascript" src="http://modernizr.com/
downloads/modernizr-latest.js"></script>
<script type="text/javascript" src="hideform.js"></script>
```

3. Next come the two buttons that will allow us to show or hide the registration form, as appropriate. Add the following code immediately below the opening `<body>` tag:

```
<section id="container">
  <button id="hide-button">Hide</button>
  <button id="show-button">Show</button>
</section>
```

4. Next is the real meat of our form. Add the following immediately below the closing `</section>` tag:

```
  <div id="sign-up-form">
    <form>
      <h3>Enter your details to sign up to our
      newsletter:</h3>
      <div class="input-prepend">
        <span class="add-on">@</span><input name="email"
        type="email" placeholder="Enter your e-mail
        address">
      </div>
      <div class="input-prepend">
        <span class="add-on">@</span><input
        name="emailconfirm" type="email" type="text"
        placeholder="Please re-enter it to confirm">
      </div>
      <input type="submit" value="Submit" name="subscribe"
      class="btn btn-small">
```

```
    </form>
  </div>
<section>
```

5. We now have a basic form in place, but it really doesn't look particularly stylish. Let's fix that by adding some styles, beginning with the basic ones to style the document:

```
body { font-size: 13px; line-height: 18px; font-weight: normal; }
section { display:block; }
form { margin:0 0 18px; }
h3 { text-rendering: optimizelegibility; margin-bottom: 7px;
margin-top: 0px; font-size: 18px; font-family: 'Shadows Into
Light', cursive;}
```

6. Next comes the styling for the input fields and submit button:

```
input, button { font-family: "Helvetica Neue", Helvetica, Arial,
sans-serif; margin-left:0; }

input[type=submit] { width:auto; }
input[type=email] { background-color: #FFFFFF; border: 1px solid
#CCCCCC; box-shadow: 0 1px 1px rgba(0,0,0,0.075) inset;
color: #555555; display: inline-block; padding: 5px; width: 89.5%;
}
```

7. We will need to provide some styles for our form too, so go ahead and add the following code:

```
#container { width:400px; margin: 100px auto 0; }
#sign-up-form { background-color: #F5F5F5; border: 1px solid
rgba(0, 0, 0, 0.05); border-radius: 4px 4px 4px 4px;
box-shadow: 0 1px 1px rgba(0, 0, 0, 0.05) inset; margin-bottom:
20px; min-height: 20px; padding: 19px; width: 375px;}
```

8. Next comes the styling for our button:

```
.btn { background: #F5F5F5 url(none) repeat-x; border-radius: 4px
4px 4px 4px; color: #333333; cursor: pointer; display: inline-
block;
box-shadow: 0 1px 0 rgba(255,255,255,0.2) inset, 0 1px 2px
rgba(0,0,0,0.05); margin-bottom: 0; text-align:center; padding:
4px 10px;
text-shadow: 0 1px 1px rgba(255,255,255,0.75); vertical-align:
middle; border: 1px solid #B3B3B3 #E6E6E6 #E6E6E6; }
```

9. Finally we need to add styles for the @ sign that prepends each input field:

```
.input-prepend { margin-bottom:5px; }
.input-prepend .add-on { margin-right: -1px; border-radius: 0 0 0
0; margin-left: -1px; background-color: #EEEEEE; border: 1px solid
#CCCCCC;
```

```
display:inline-block; height: 18px; min-width: 16px; text-align:
center; text-shadow: 0 1px 0 #FFFFFF;
vertical-align: middle; width: auto; padding: 4px 5px; border-
radius: 3px 0 0 3px; }
.input-prepend input { border-radius: 0 3px 3px 0; margin-bottom:
0; position: relative; vertical-align: middle; }
```

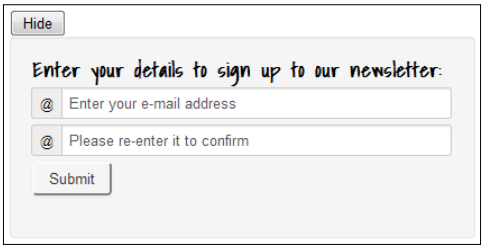
10. Let's now add the script that will add the Local Storage functionality, and make the form work, beginning with a check for Modernizr, followed by the event handler for the Hide button:

```
$(document).ready(function($){
  if (Modernizr.localstorage) {
    $('#hide-button').click(function(e){
      localStorage.setItem('subscribed',true);
      $('#sign-up-form,#hide-button').slideToggle();
      $('#hide-button').hide();
      $('#show-button').show();
    });
```

11. We pair that with the event handler for the Show button and then show or hide the buttons accordingly, depending on whether we have already subscribed or not:

```
  $('#show-button').click(function(e){
    localStorage.setItem('subscribed', false);
    localStorage.clear();
    $('#sign-up-form').slideToggle();
    $('#hide-button').show();
    $('#show-button').hide();
  });
  var is_subscribed = localStorage.getItem('subscribed');
  if(is_subscribed){
    $('#sign-up-form,#hide-button').slideToggle();
  }
  if(!is_subscribed){
    $('#sign-up-form').show();
    $('#show-button').hide();
  }
}
});
```

12. If all is well, you should see the following result when previewing this in a browser. This has the subscribed value already set:



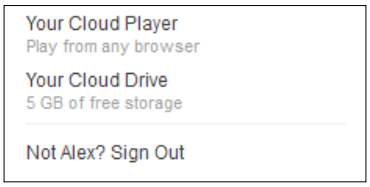
How it works...

This recipe works on the principle that we control which of the two buttons are visible, depending on the value set for `subscribed` in `LocalStorage`.

By default, when you first visit the form, you will see both. It is only when you've submitted your details for the first time that it sets the `subscribed` in `LocalStorage` to `true`. On subsequent visits, the code checks for this first and then shows or hides the form and hide button, depending on whether or not it can retrieve a value for the `subscribed` attribute from Local Storage.

There's more...

This is a very simplified example. It is likely that you would not want to have both buttons being shown, but set something more akin to what Amazon does on its website:



Notice the **Not Alex? Sign Out** option at the bottom of the menu. This is a perfect example of how you could use Local Storage. If a user has signed in, you could store their details in Local Storage, and then use a menu option such as the one in the preceding screenshot to display the login form so that the user can either log in with their own details or register as a new user.

We have now come to the end of the book. We looked at a number of recipes to show you how you can use the power of Local Storage within your pages. This is only just the start of what you can achieve using the functionality. There is a whole world out there to explore, which is only limited by the bounds of your imagination. I hope you've enjoyed working through the recipes, just as much as I have enjoyed writing this book!

Thank you for buying
Instant HTML5 Local Storage How-to

About Packt Publishing

Packt, pronounced 'packed', published its first book "*Mastering phpMyAdmin for Effective MySQL Management*" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

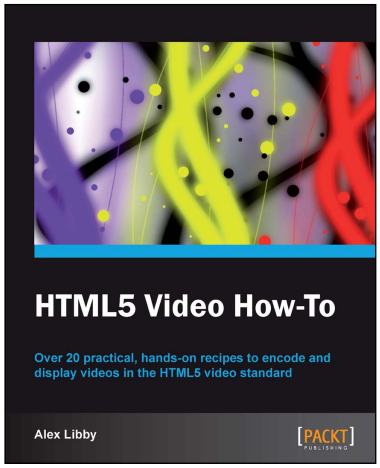
Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website: `www.packtpub.com`.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to `author@packtpub.com`. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.

[PACKT] PUBLISHING



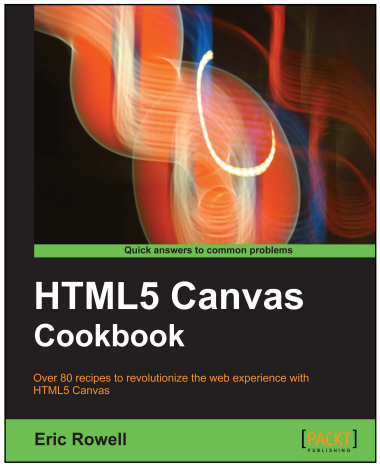
HTML5 Video How-To

ISBN: 978-1-84969-364-6 Paperback: 320 pages

Over 20 practical, hands-on recipes to encode and display videos in the HTML5 video standard

1. Learn something new in an Instant! A short, fast, focused guide delivering immediate results.
2. Encode and embed videos into web pages using the HTML5 video standard
3. Publish videos to popular sites, such as YouTube or VideoBin
4. Provide cross-browser support for HTML5 videos and create your own custom video player using jQuery

---



HTML5 Canvas Cookbook

ISBN: 978-1-84969-136-9 Paperback: 348 pages

Over 80 recipes to revolutionize the web experience with HTML5 Canvas

1. The quickest way to get up to speed with HTML5 Canvas application and game development
2. Create stunning 3D visualizations and games without Flash
3. Written in a modern, unobtrusive, and objected oriented JavaScript style so that the code can be reused in your own applications.
4. Part of Packt's Cookbook series: Each recipe is a carefully organized sequence of instructions to complete the task as efficiently as possible