9

Useful jQuery Recipes for ASP.NET Sites

We have reached toward the end of the exciting journey of exploring the rich features of jQuery with ASP.NET applications. This chapter summarizes the concepts we have covered so far in the following recipes:

- Infinite scrolling
- Creating a shadow effect for text
- Using Ajax to load scripts in web pages
- Serializing form data
- Uploading files in MVC
- Exporting the GridView data in the CSV format

Introduction

This book began with the fundamentals of jQuery, such as the use of selectors, event handling, and DOM manipulation. After learning the basics, we moved on to creating visual effects using graphics and animations. We then worked with Ajax and created plugins.

It is possible to apply more than one feature of jQuery on a single page. This chapter describes six diverse recipes that can be embedded in real-world applications to solve common problems.

Infinite scrolling

Sites such as Facebook and Twitter provide auto loading of contents when the end of the page is reached on scrolling. This is referred to as **infinite loading**. This recipe will demonstrate the use of jQuery to implement this feature on a data-driven page.

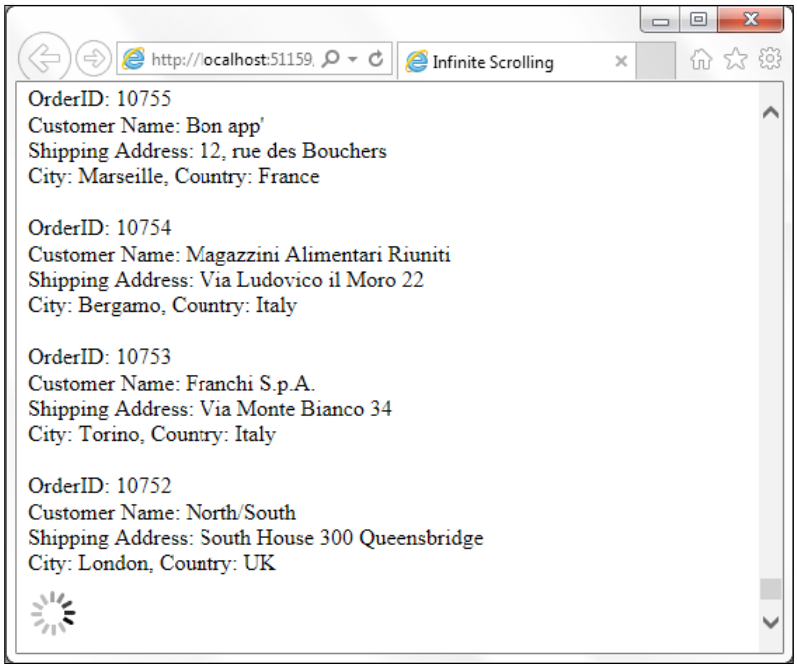
The constructs used in this example are summarized in the following table:

Construct	Type	Description
`$("#identifier")`	jQuery selector	This selects an element based on its ID
`$.ajax()`	jQuery function	This posts an Ajax request to the server with the set options
`.append()`	jQuery method	This inserts content at the end of each matched element
`.height()`	jQuery method	This gets the height of the first matched element or sets the height of each matched element
`.hide()`	jQuery method	This hides the matched elements
`.length`	jQuery property	This returns the number of elements in the jQuery object
`.scroll()`	jQuery event binder	This attaches an event handler for the scroll event of the matched elements
`.scrollTop()`	jQuery method	This gets the vertical position of the scrollbar for the first matched element or sets the vertical position of the scrollbar for each matched element
`this`	DOM element	This refers to the current DOM element
`window.location.href`	JavaScript property	This returns the URL of the current page

Getting ready

To add infinite scrolling to a page, follow these steps:

1. In this example, we will display the order details from the Northwind database on the web form. For each database read, a fixed number of records will be retrieved and displayed. A loader image will be displayed at the end of the current set of records, as shown in the following screenshot:



When the scroll bar reaches the bottom of the page, the next set of records will be auto loaded and displayed.

2. To create this form, launch a new **ASP.NET Web Application** project in Visual Studio using the **Empty** template and name it `Recipe1` (or any other suitable name).
3. Add a `Scripts` folder to the project and include the jQuery files in this folder.
4. Add a web form named `Default.aspx` to the project. Include the jQuery library in the form.

5. Next, download a loader image from `http://www.ajaxload.info`:

   For the **Indicator type** field, select **Indicator Big**, and click on the **Generate It !** button, as shown in the preceding screenshot. Save the image as `ajax-loader.gif` in the `images` folder.

6. Add the following markup to the form:

```
<div id="container"></div>
<asp:Image ID="imgLoad" runat="server"
ImageUrl="~/images/ajax-loader.gif" />
```

7. Next, we will add a class to the order details. To do this, right-click on the project, and go to **Add** | **Class**. Name the class as `Orders.vb` (VB) or `Orders.cs` (C#). Include the following properties in the class.

   For VB, the properties are as follows:

```
Public Class Orders
  Public Property OrderID
  Public Property ShipName
  Public Property ShipAddress
  Public Property ShipCity
  Public Property ShipCountry
End Class
```

For C#, the properties are as follows:

```
public class Orders
{
  public string OrderID { get; set; }
  public string ShipName { get; set; }
  public string ShipAddress { get; set; }
  public string ShipCity { get; set; }
  public string ShipCountry { get; set; }
}
```

8. In the `web.config` file, add a connection string to the Northwind database in the configuration section:

```
<connectionStrings>
  <add name="NorthwindConnection"
providerName="System.Data.SqlClient" connectionString="Data
Source=localhost;Initial Catalog=Northwind;Integrated
Security=True;"/>
</connectionStrings>
```

9. Now, we will add a page method to the code-behind of the web form to retrieve the order details. Let's start by adding the following namespaces at the top of the page.

   For VB, the namespace is as follows:

```
Imports System.Web.Services
Imports System.Data
Imports System.Data.SqlClient
Imports System.Web.Configuration
```

   For C#, the namespace is as follows:

```
using System.Web.Services;
using System.Data;
using System.Data.SqlClient;
using System.Web.Configuration;
```

10. Let's define two `shared` (VB)/`static` (C#) variables. The `currentIndex` variable keeps track of the current index of the order details' records being displayed, whereas the `pageSize` variable keeps track of the total number of records per page.

    For VB, the definition is as follows:

```
Shared currentIndex As Integer
Shared pageSize As Integer
```

For C#, the definition is as follows:

```
static int currentIndex, pageSize;
```

Initialize these variables in the page load procedure. We will use a page size of 10 records.

For VB, the code is as follows:

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Load
  currentIndex = 0
  pageSize = 10
End Sub
```

For C#, the code is as follows:

```
protected void Page_Load(object sender, EventArgs e)
{
  currentIndex = 0;
  pageSize = 10;
}
```

11. Add the following page method to the code-behind. This method will retrieve the current set of records using ADO.NET.

For VB, the code is as follows:

```
<WebMethod()>
Public Shared Function GetMoreOrders() As Orders()
  Dim orderList As List(Of Orders) = New List(Of Orders)()
  Dim strConn As String =
WebConfigurationManager.ConnectionStrings
("NorthwindConnection").ConnectionString
  Dim con As SqlConnection = New SqlConnection(strConn)
  Dim strSql As String = "select OrderID, ShipName,
ShipAddress, ShipCity, ShipCountry from orders order by
OrderID desc"
  con.Open()
  Dim adapter As SqlDataAdapter = New
SqlDataAdapter(strSql, strConn)
  Dim ds As DataSet = New DataSet
  adapter.Fill(ds, currentIndex, pageSize, "Orders")
  For Each dr In ds.Tables("Orders").Rows
    Dim orderObj As New Orders()
    orderObj.OrderID = Convert.ToString(dr("OrderID"))
    orderObj.ShipName = Convert.ToString(dr("ShipName"))
    orderObj.ShipAddress =
Convert.ToString(dr("ShipAddress"))
```

```
    orderObj.ShipCity = Convert.ToString(dr("ShipCity"))
    orderObj.ShipCountry =
Convert.ToString(dr("ShipCountry"))
    orderList.Add(orderObj)
  Next
  con.Close()
  currentIndex = currentIndex + pageSize
  Return orderList.ToArray
End Function
```

For C#, the code is as follows:

```
[WebMethod]
public static Orders[] GetMoreOrders()
{
  List<Orders> orderList = new List<Orders>();
  string strConn =
WebConfigurationManager.ConnectionStrings
["NorthwindConnection"].ConnectionString;
  SqlConnection con = new SqlConnection(strConn);
  string strSql = "select OrderID, ShipName, ShipAddress,
ShipCity, ShipCountry from orders order by OrderID desc";
  con.Open();
  SqlDataAdapter adapter = new SqlDataAdapter(strSql,
strConn);
  DataSet ds = new DataSet();
  adapter.Fill(ds, currentIndex, pageSize, "Orders");
  foreach (DataRow dr in ds.Tables["Orders"].Rows)
  {
    orderList.Add(new Orders { OrderID =
Convert.ToString(dr["OrderID"]), ShipName =
Convert.ToString(dr["ShipName"]),
ShipAddress=Convert.ToString(dr["ShipAddress"]), ShipCity =
Convert.ToString(dr["ShipCity"]), ShipCountry =
Convert.ToString(dr["ShipCountry"]) });
  }
  con.Close();
  currentIndex += pageSize;
  return orderList.ToArray();
}
```

The preceding page method returns an array of objects of the `Orders` type to the calling script. The Northwind connection string in `web.config` is used to connect to the database using ADO.NET. A data adapter is used to populate the dataset with records, starting from `currentIndex`. The number of records returned during each query is equal to the `pageSize` variable. The data rows in the returned dataset is looped and stored in a `List` object. This `List` object is converted to an array and returned to the calling script. The `currentIndex` variable is then incremented by the `pageSize` variable to get the starting position of the next set of records.

Note that we are using Windows Authentication for all database driven examples in this book. Hence in the MS SQL Server, it is important to give permission to the Windows account to access the Northwind database.

How to do it...

Add the following jQuery code to a `script` block on the page:

```
<script type="text/javascript">
  $(document).ready(function() {
    loadNextPage();
    $(window).scroll(function() {
      var scrolldist = $(window).scrollTop() + $(window).height();
      if ($(document).height() == scrolldist)
        loadNextPage();
    });
    function loadNextPage() {
      var loc = window.location.href;
      $.ajax({
        type: "POST",
        url: loc + "/GetMoreOrders",
        data: '{}',
        contentType: "application/json;charset=utf-8",
        dataType: "json",
        timeout: 5000,
        cache: false,
        success: function(response) {
          if (response.d.length > 0) {
            $.each(response.d, function() {
              $("#container").append("OrderID: " + this['OrderID']
+ "<br/>");
              $("#container").append("Customer Name: " +
this['ShipName'] + "<br/>");
```

```
                $("#container").append("Shipping Address: " +
this['ShipAddress'] + "<br/>");
                $("#container").append("City: " + this['ShipCity'] +
", Country: " + this['ShipCountry'] + "<br/><br/>");
              });
            } else
              $("#<%=imgLoad.ClientID%>").hide();
          },
          error: function(jqXHR, textStatus, errorThrown) {
            if (textStatus == "error") {
              alert("An error has occurred: " + jqXHR.status + " " +
jqXHR.statusText);
            }
          }
        });
      }
    });
</script>
```

How it works...

The infinite scrolling of the page works as follows:

1. In the jQuery script, when the document is ready, the `loadNextPage()` function is called. This function initializes the page content by displaying an initial set of 10 records from the database.

   In the `loadNextPage()` function, we first get the URL of the current page:

   ```
   var loc = window.location.href;
   ```

   An Ajax call is made with the following parameters:

   - The type of the request is set to `POST`:

     ```
     type: "POST",
     ```

   - The `url` of the request is set to `UrlOfCurrentPage/NameOfPageMethod`:

     ```
     url: loc + "/GetMoreOrders",
     ```

   - No data is sent to the server:

     ```
     data: '{}',
     ```

   - The `contentType` of the request is set to `application/json` and the character is set to `utf-8`:

     ```
     contentType: "application/json;charset=utf-8",
     ```

- The `dataType` of the response is set to `json`:

  ```
  dataType: "json",
  ```

- A `timeout` of `5000` milliseconds is set after which the request will timeout if no response is received from the server:

  ```
  timeout: 5000,
  ```

- The cache is set to `false` so that the response is not cached:

  ```
  cache: false,
  ```

- A callback function is defined for the successful completion of the Ajax call:

  ```
  success: function (response) {…},
  ```

If the response length is nonzero, we loop through the returned array and append the data to the `container` div on the page:

```
if (response.d.length > 0) {
  $.each(response.d, function () {
    $("#container").append("OrderID: " + this['OrderID'] +
"<br/>");
    $("#container").append("Customer Name: " +
this['ShipName'] + "<br/>");
    $("#container").append("Shipping Address: " +
this['ShipAddress'] + "<br/>");
    $("#container").append("City: " + this['ShipCity'] + ",
Country: " + this['ShipCountry'] + "<br/><br/>");
  });
}
```

If the response is empty, then the loader image is hidden. This indicates that all the records have been completely loaded:

```
$("#<%=imgLoad.ClientID%>").hide();
```

- A callback function is defined when the request is unsuccessful:

```
error: function (jqXHR, textStatus, errorThrown) {…}
```

If the `textStatus` returns `error`, then the `status` and `statusText` of the `XmlHttpRequest` object is displayed:

```
if (textStatus == "error") {
  alert("An error has occurred: " + jqXHR.status + " " +
jqXHR.statusText);
}
```

2. After the page content is loaded for the first time, an event handler is attached to the `scroll` event of the window:

   ```
   $(window).scroll(function () {…});
   ```

   In the preceding event handler, to determine whether the scroll bar has reached the bottom of the browser window, we determine how much of the document has already been scrolled:

   ```
   var scrolldist = $(window).scrollTop() +
   $(window).height();
   ```

   Next, if the document height becomes equal to the value computed above, that is, the scroll bar has reached the end of the document, we call the `loadNextPage()` method to get the next set of records from the database:

   ```
   if ($(document).height() == scrolldist)
     loadNextPage();
   ```

   The previous checks are performed every time the user scrolls the window.

`$(window).height()` returns the height of the browser viewport, that is, the visible area of the HTML content.

`$(document).height()` returns the height of the entire HTML page.

See also

The *Using Ajax to load scripts in web pages* recipe

Creating a shadow effect for text

This recipe demonstrates how to create a shadow effect on text elements with the help of CSS properties. The constructs used in this example are summarized in the following table:

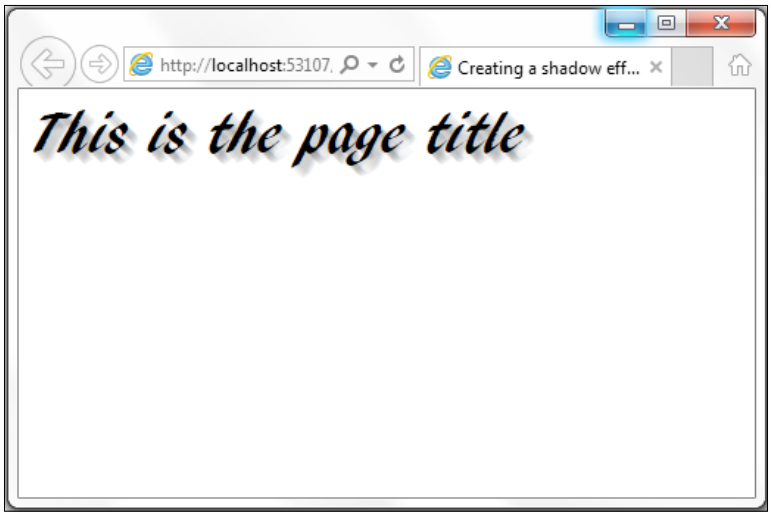
Construct	Type	Description
`$("#identifier")`	jQuery selector	This selects an element based on its ID.
`.appendTo()`	jQuery method	This inserts elements at the end of the target.

Construct	Type	Description
`.clone()`	jQuery method	This makes a deep copy of the matched elements, that is, the matched elements are copied along with their descendants and text nodes.
`.css()`	jQuery method	This gets the style property for the first matched element or sets the style property for every matched element.
`left`	CSS property	This is the position of the left edge of the element.
`.offset()`	jQuery method	This gets the coordinates of the first matched element relative to the document. It returns an object with the left and top properties for the left and top coordinates of the element, respectively.
`opacity`	CSS property	This is the degree of transparency of the element.
`position`	CSS property	This specifies the position of an element as fixed, relative, absolute, or sticky.
`top`	CSS property	This is the position of the top edge of the element.
`zIndex`	CSS property	This is the z-order of an element. When elements overlap, the one with the higher z-order appears above the one with the lower z-order.

Getting ready

To create a shadow effect on text content, follow these steps:

1. We will create a shadow effect on text elements, as shown in the following screenshot:



2. To get started, launch a new **ASP.NET Web Application** project in Visual Studio using the **Empty** template and name it `Recipe2` (or any other suitable name).
3. Include the jQuery library in a `Scripts` folder in the project.
4. Add a web form named `Default.aspx` and include the jQuery library in the form.
5. Add a `div` element with some text, as shown in the following code:

```
<div id="title">
  This is the page title
</div>
```

6. To style the `title` text, add the following CSS:

```
<style type="text/css">
  #title{
    font-family:'AR BLANCA';
    font-size:40px;
  }
</style>
```

How to do it...

Add the following jQuery code to a `script` block on the form:

```
<script type="text/javascript">
  $(document).ready(function() {
    var $originalLeft = $("#title").offset().left;
    var $originalTop = $("#title").offset().top;
    for (var cnt = 0; cnt < 7; ++cnt) {
      var $clonedItem = $("#title").clone();
      $clonedItem.css({
        opacity: 0.07,
        left: $originalLeft + cnt + 1,
        top: $originalTop + cnt + 1,
        zIndex: -1,
        position: 'absolute'
      }).appendTo("body");
    }
  });
</script>
```

How it works...

The shadow effect works as follows:

1. When the document is ready, get the left and top coordinates of the element on which the shadow effect is to be applied:

   ```
   var $originalLeft = $("#title").offset().left;
   var $originalTop = $("#title").offset().top;
   ```

2. The shadow will be formed by cloning the text element a couple of times and shifting the position of the cloned text with respect to the original element. Let's say that we clone the element seven times (determined randomly from experiment). Thus, we run a `for` loop the required number of times, as shown in the following code:

   ```
   for (var cnt = 0; cnt < 7; ++cnt) {…}
   ```

   Inside the preceding `for` loop, first of all, clone the original element:

   ```
   var $clonedItem = $("#title").clone();
   ```

   Set the CSS of the cloned element, as follows:

   - Set its opacity to `0.07`
   - Shift the left edge of the clone with respect to the left edge of the original element by `cnt + 1`

- Shift the top edge of the clone with respect to the top edge of the original element by `cnt + 1`
- Set the `zIndex` value of the clone to `-1` so that it is stacked below the original element
- Set the position of the clone to absolute:

```
$clonedItem.css({
   opacity: 0.07,
   left: $originalLeft + cnt + 1,
   top: $originalTop + cnt + 1,
   zIndex: -1,
   position: 'absolute'
})
```

The cloned element is then appended to the `body` element:

```
.appendTo("body");
```

See also

The *Serializing form data* recipe

Using Ajax to load scripts in web pages

This recipe demonstrates lazy loading of script files, that is, delaying the loading of scripts until they are needed at runtime. We will use the jQuery cycle plugin to demonstrate this.

The constructs used in this example are summarized as follows:

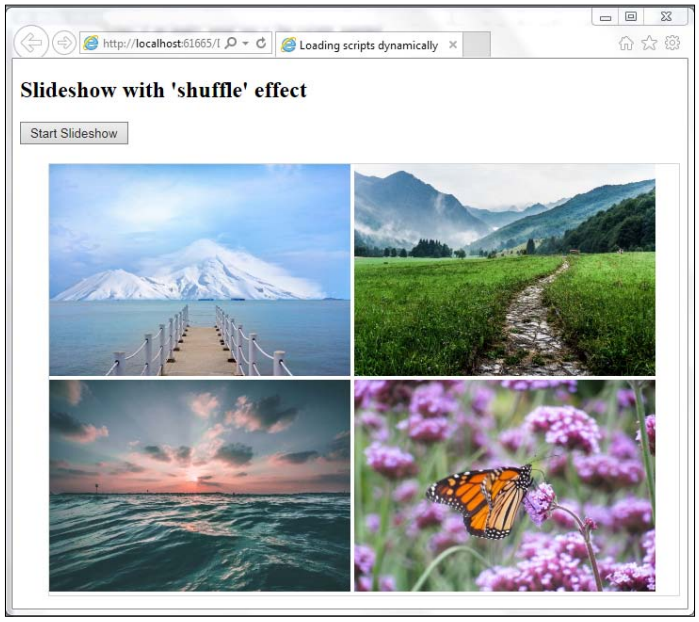
Construct	Type	Description
`$("#identifier")`	jQuery selector	This selects an element based on its ID.
`$(this)`	jQuery object	This refers to the current jQuery object.
`$.getScript()`	jQuery function	This loads a JavaScript file from the server using HTTP `GET` and executes it.
`click`	jQuery event	This is fired when you click on an element. It corresponds to the JavaScript `click` event.
`.cycle()`	jQuery cycle plugin method	This runs a slideshow on the child elements of the matched element.
`event.preventDefault()`	jQuery method	This prevents the default action of the event from being triggered.
`.on()`	jQuery event binder	This attaches an event handler for one or more events to the matched elements.

Construct	Type	Description
`.prop(propertyName)` or `.prop(propertyName, value)`	jQuery method	This returns the value of the specified property for the first matched element or sets the value of the specified property for all matched elements.

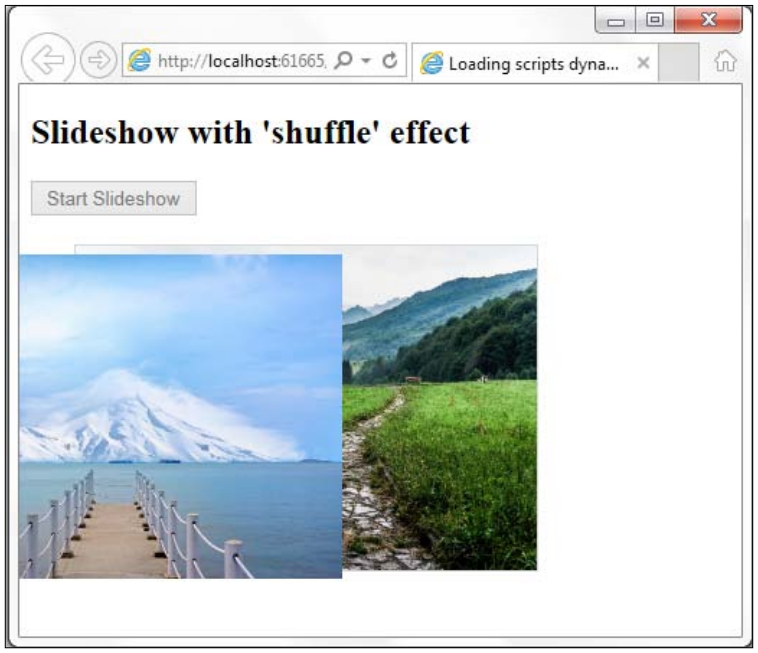
Getting ready

Follow these steps to build a page with lazy loading of scripts:

1. Let's create a page with a few sample images, as shown in the following screenshot:



After clicking on the **Start Slideshow** button, the cycle plugin will be dynamically loaded using jQuery. This will trigger the slideshow on the images, as shown in the following screenshot:



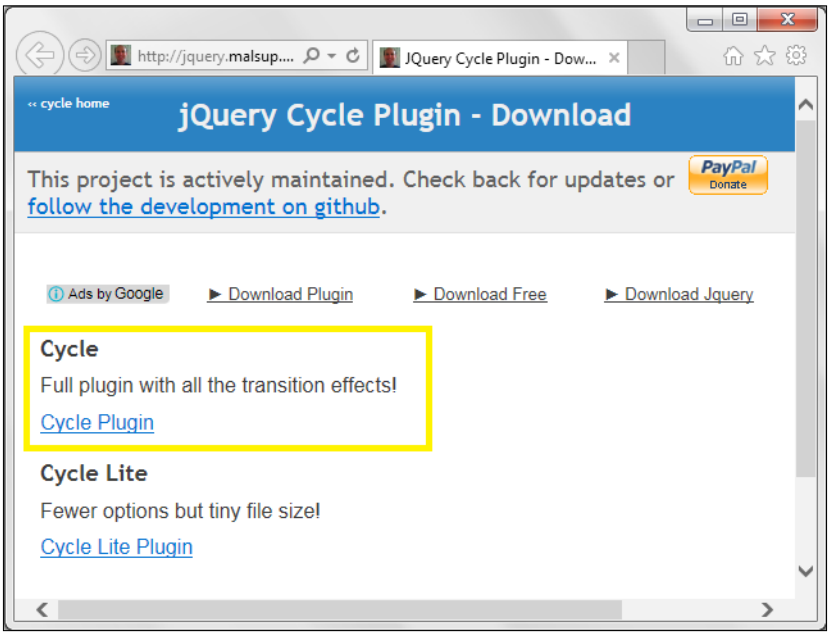
2. To build this page, launch a new **ASP.NET Web Application** project in Visual Studio using the **Empty** template and name it `Recipe3` (or any other suitable name).
3. Add the jQuery library to the `Scripts` folder in the project.
4. Create an `images` folder and add some sample images to this folder.
5. Add a web form named `Default.aspx` to the project. Include the jQuery library in the form.
6. Add the following markup to the page:

```
<asp:Button ID="btnStart" runat="server" Text="Start
Slideshow" />
<div id="container">
  <asp:Image ID="imgDemo1" runat="server"
ImageUrl="~/images/image1.jpg"/>
  <asp:Image ID="imgDemo2" runat="server"
ImageUrl="~/images/image2.jpg"/>
  <asp:Image ID="imgDemo3" runat="server"
ImageUrl="~/images/image3.jpg" />
  <asp:Image ID="imgDemo4" runat="server"
ImageUrl="~/images/image4.jpg" />
</div>
```

7. Include the following styles for the images and the `container div`:

```
<style type="text/css">
  img{
    width:320px;
    height:225px;
  }
  #container{
    margin-left:30px;
    margin-top:20px;
    border:solid;
    border-color:lightgrey;
    border-width:1px;
  }
</style>
```

8. Download the cycle plugin from `http://jquery.malsup.com/cycle/download.html`:



Save the `jquery.cycle.all.js` plugin file in the `Scripts` folder.

How to do it...

Add the following jQuery code to a `script` block on the page:

```
<script type="text/javascript">
  $(document).ready(function() {
    $("#<%=btnStart.ClientID%>").on("click", function(evt) {
      evt.preventDefault();
      $(this).prop("disabled", true);
      var url = "Scripts/jquery.cycle.all.js";
      $.getScript(url)
        .done(function(script, textStatus) {
          $("#container").cycle({
            fx: "shuffle",
            speed: 1000,
            timeout: 1000
          });
        })
        .fail(function(jqxhr, settings, exception) {
          alert("Failed to load script");
        });
    });
  });
</script>
```

How it works...

The lazy loading of the cycle plugin works as follows:

1. The cycle plugin will be loaded on the page after clicking on the **Start Slideshow** button. Hence, we write an event handler for the button `click` event, as follows:

   ```
   $("#<%=btnStart.ClientID%>").on("click", function (evt)
   {…});
   ```

2. In the preceding event handler, first, prevent the page from auto postback:

   ```
   evt.preventDefault();
   ```

   Next, disable the button so that it is not clickable:

   ```
   $(this).prop("disabled", true);
   ```

   Initialize the URL of the cycle plugin:

   ```
   var url = "Scripts/jquery.cycle.all.js";
   ```

Make an HTTP `GET` request to retrieve the cycle plugin from the server:

```
$.getScript(url)
```

When the loading of the cycle plugin is completed successfully, the following function will be executed:

```
.done(function (script, textStatus) {…});
```

In the preceding function, trigger the slideshow by providing the following options to the plugin:

- Use the `shuffle` transition effect. A number of other transition effects, such as fade, zoom, turndown, scrollRight, and so on, can also be used.
- The `speed` of the transition is set to `1000` milliseconds.
- Each slide will be displayed for `1000` milliseconds using the `timeout` option.

The preceding options are set as follows:

```
$("#container").cycle({
  fx: "shuffle",
  speed: 1000,
  timeout: 1000
});
```

If, however, the loading of the cycle plugin fails, display the required error message to the user:

```
.fail(function (jqxhr, settings, exception) {
  alert("Failed to load script");
});
```

The `$.getScript()` function is a shorthand function for the following Ajax command:

```
$.ajax({  type: "GET",
  url: url,
  cache: false,
  dataType: "script",
  ...
});
```

By default, the `$.getScript()` function sets cache to `false`. This can, however, be overwritten using `$.ajaxSetup()`.

See also

The *Infinite scrolling* recipe

Serializing form data

Serializing form data is the process of converting the form fields into name/value pairs. In this recipe, let's serialize form data and send it to the server using Ajax. The constructs used in this example are summarized as follows:

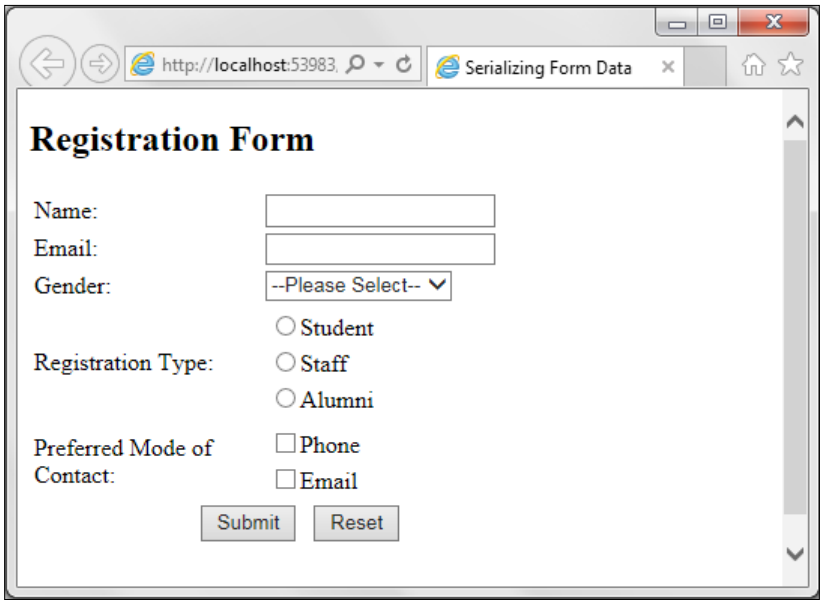
Construct	Type	Description
`$("#identifier")`	jQuery selector	This selects an element based on its ID.
`$.ajax()`	jQuery function	This posts an Ajax request to the server with the set options.
`:checked`	jQuery filter	This selects checked input elements.
`click`	jQuery event	This is fired when you click on an element. It corresponds to the JavaScript `click` event.
`event.preventDefault()`	jQuery method	This prevents the default action of the event from being triggered.
`.find()`	jQuery method	This finds all elements that match the filter.
`.hide()`	jQuery method	This hides the matched elements.
`.html()`	jQuery method	This returns the HTML content of the first matched element or sets the HTML content of every matched element.
`JSON.stringify()`	JavaScript function	This converts a JavaScript value to a JSON string.
`.not()`	jQuery method	This removes elements from the matched elements.
`.on()`	jQuery event binder	This attaches an event handler for one or more events to the matched elements.

Construct	Type	Description
`.prop(propertyName)` or `.prop(propertyName, value)`	jQuery method	This returns the value of the specified property for the first matched element or sets the value of the specified property for all matched elements.
`.serializeArray()`	jQuery method	This encodes form elements as an array of name/value pairs.
`.show()`	jQuery method	This displays the matched elements.
`.val()`	jQuery method	This returns the value of first matched element or sets the value of every matched element.
`window.location.href`	JavaScript property	This returns the URL of the current page.

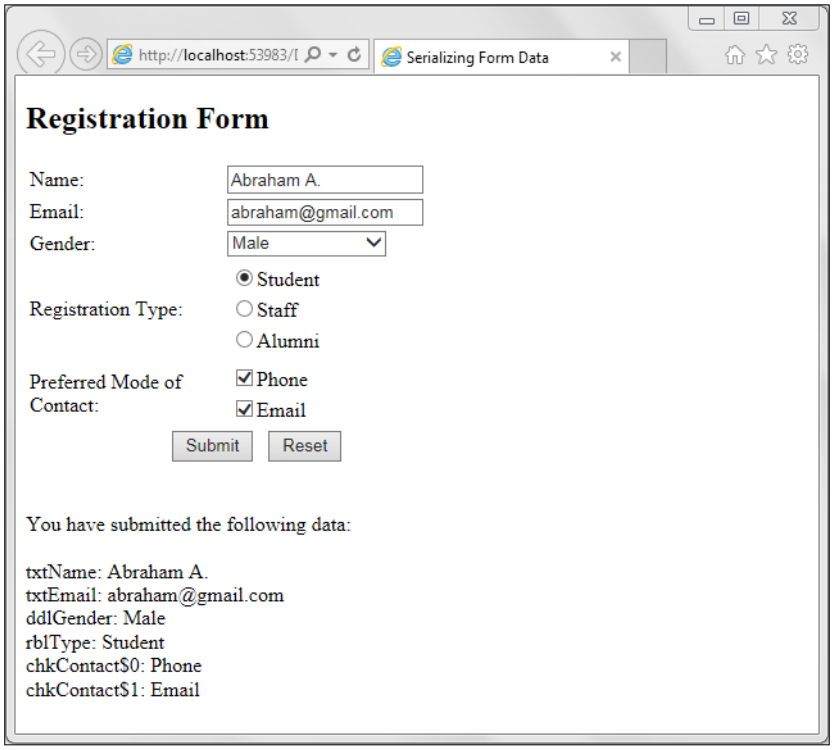
Getting ready

Follow these steps to create a page that serializes form data:

1. In this example, let's create a simple registration page consisting of the following fields:



When the form is submitted, the form data is serialized and posted to the server using Ajax. At the server, we will parse this data and return it back as an Ajax response. This will be displayed on the browser, as shown in the following screenshot. Notice that the name/value pairs consist of the control names and the corresponding data entered.



2. To get started, create a new **ASP.NET Web Application** project in Visual Studio using the **Empty** template and name it `Recipe4` (or any other suitable name).
3. Add a `Scripts` folder to the project and include the jQuery library in this folder.
4. Add a web form and name it `Default.aspx`. Include the jQuery library in the form.
5. To create the form fields, add the following markup to the form:

```
<div>
  <table>
    <tr>
      <td class="col1">
        <asp:Label ID="lblName" runat="server"
Text="Name:"></asp:Label>
      </td>
      <td class="col2">
```

```
        <asp:TextBox ID="txtName"
runat="server"></asp:TextBox>
      </td>
    </tr>
    <tr>
      <td>
        <asp:Label ID="lblEmail" runat="server"
Text="Email:"></asp:Label>
      </td>
      <td>
        <asp:TextBox ID="txtEmail"
runat="server"></asp:TextBox>
      </td>
    </tr>
    <tr>
      <td>
        <asp:Label ID="lblGender" runat="server"
Text="Gender:"></asp:Label>
      </td>
      <td>
        <asp:DropDownList ID="ddlGender" runat="server">
          <asp:ListItem Text="--Please Select--"
Value=""></asp:ListItem>
          <asp:ListItem Text="Male"
Value="Male"></asp:ListItem>
          <asp:ListItem Text="Female"
Value="Female"></asp:ListItem>
        </asp:DropDownList>
      </td>
    </tr>
    <tr>
      <td>
        <asp:Label ID="lblType" runat="server"
Text="Registration Type:"></asp:Label>
      </td>
      <td>
        <asp:RadioButtonList ID="rblType" runat="server">
          <asp:ListItem Text="Student">Student</asp:ListItem>
          <asp:ListItem Text="Staff">Staff</asp:ListItem>
          <asp:ListItem Text="Alumni">Alumni</asp:ListItem>
        </asp:RadioButtonList>
      </td>
    </tr>
    <tr>
      <td>
```

```
        <asp:Label ID="lblContact" runat="server"
Text="Preferred Mode of Contact:"></asp:Label>
      </td>
      <td>
        <asp:CheckBoxList ID="chkContact" runat="server">
          <asp:ListItem Text="Phone"
Value="Phone"></asp:ListItem>
          <asp:ListItem Text="Email"
Value="Email"></asp:ListItem>
        </asp:CheckBoxList>
      </td>
    </tr>
    <tr>
      <td colspan="2" class="center">
        <asp:Button ID="btnSubmit" runat="server"
Text="Submit" />  
        <asp:Button ID="btnReset" runat="server"
Text="Reset" />
      </td>
    </tr>
  </table>
  <br /><br />
  <asp:Panel ID="pnlResult" runat="server"></asp:Panel>
</div>
```

Note that a `pnlResult` Panel control is included at the end of the form. This Panel control will be used to display the Ajax response.

6. Include the following styles for the form fields:

```
<style type="text/css">
  .col1{
    width:150px;
  }
  .col2{
    width:200px;
  }
  .center{
    text-align:center;
  }
</style>
```

7. Next, we will add a class for the name/value pairs generated by serializing the form fields. The name refers to the field name, whereas the value refers to the data entered by the user in the respective field.

To add the class, right-click on the project in the **Solution Explorer** tab, and go to **Add | Class**. Name the class as `NameValuePairs`.

For VB, the code is as follows:

```
Public Class NameValuePairs
  Public Property Name
  Public Property Value
End Class
```

For C#, the code is as follows:

```
public class NameValuePairs
{
  public string Name { get; set; }
  public string Value { get; set; }
}
```

8. In the code-behind page of the web form, add the `System.Web.Services` namespace at the top of the page.

   For VB, the namespace is as follows:

   ```
   Imports System.Web.Services
   ```

   For C#, the namespace is as follows:

   ```
   using System.Web.Services;
   ```

9. Add a page method to the code-behind. The serialized form data will be posted to this method.

   For VB, the method is as follows:

   ```
   <WebMethod>
   Public Shared Function ProcessForm(formData As
   NameValuePairs()) As String
     Dim strReturn As String = String.Empty
     If (formData.Length > 0) Then
       strReturn += "You have submitted the following data:
   <br/><br/>"
       For Each item As NameValuePairs In formData
         strReturn += item.Name + ": " + item.Value + "<br/>"
       Next
     End If
     Return strReturn
   End Function
   ```

For C#, the method is as follows:

```
[WebMethod]
public static string ProcessForm(NameValuePairs[] formData)
{
  string strReturn = String.Empty;
  if (formData.Length > 0) {
    strReturn += "You have submitted the following data:
<br/><br/>";
    foreach ( NameValuePairs item in formData)
      strReturn += item.Name + ": " + item.Value + "<br/>";
  }
  return strReturn;
}
```

The preceding page method receives the form data as an array of the `NameValuePairs` type. It loops through the elements of this array and builds a string of these name/value pairs. The string is returned as an Ajax response to the calling script.

How to do it...

Add the following jQuery code to a `script` block on the page:

```
<script type="text/javascript">
  $(document).ready(function() {
    $("#<%=pnlResult.ClientID%>").hide();
    $("#<%=btnSubmit.ClientID%>").on("click", function(evt) {
      evt.preventDefault();
      var strData =
$("#form1").find("input,select,textarea").not("#__VIEWSTATE").
not("#__VIEWSTATEGENERATOR").not("#__EVENTVALIDATION").
serializeArray();
      var loc = window.location.href;
      $.ajax({
        type: "POST",
        url: loc + "/ProcessForm",
        data: JSON.stringify({
          formData: strData
        }),
        dataType: "json",
        contentType: "application/json;charset=utf-8",
        timeout: 5000,
        cache: false,
        success: function(response) {
```

```
            $("#<%=pnlResult.ClientID%>").html(response.d).show();
          },
          error: function(jqXHR, textStatus, errorThrown) {
            if (textStatus == "error") {
              alert("An error has occurred: " + jqXHR.status + " " +
jqXHR.statusText);
            }
          }
        });
      });
      $("#<%=btnReset.ClientID%>").on("click", function(evt) {
        evt.preventDefault();
        $("#<%=txtName.ClientID%>").val("");
        $("#<%=txtEmail.ClientID%>").val("");
        $("#<%=ddlGender.ClientID%>").val("");
        $("#<%=rblType.ClientID%> :checked").prop("checked", false);
        $("#<%=chkContact.ClientID%> :checked").prop("checked",
false);
      });
    });
</script>
```

How it works...

The serialization of the form data works as follows:

1. When the page is loaded, the Panel control is initially hidden:

   ```
   $("#<%=pnlResult.ClientID%>").hide();
   ```

2. An event handler is attached to the `click` event of the **Submit** button, as follows:

   ```
   $("#<%=btnSubmit.ClientID%>").on("click", function (evt)
   {…});
   ```

   In this event handler, the page is prevented from posting back:

   ```
   evt.preventDefault();
   ```

   Next, the form fields are serialized and stored in an array. At runtime, ASP.NET creates additional hidden fields for `ViewState` and `EventValidation`. On viewing the page source, we can see that the `__VIEWSTATE`, `__VIEWSTATEGENERATOR`, and `__EVENTVALIDATION` fields are created, as shown in the following screenshot:

```
<div class="aspNetHidden">
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
value="hLVTpZDKuoWqmbjtbZbc1tjAoVQWw69NK2b9/EqucsE8PIQRF9W6nrh2LSkrEueYwlGfKqiv6TSveF745MSzZ
VlCqz/xI+Y2
+ydQN6EGtnbDXog6bfH3tibAhDcqWnIbwooflLMXIKuuPSvGfomUoNyrSyVrWeNHKgDRiIdGF405sPuNVvVynwHSLYdi
JCv2ii5TjcGj4ADnKima6YmskQ==" />
</div>
<div class="aspNetHidden">

      <input type="hidden" name="__VIEWSTATEGENERATOR" id="__VIEWSTATEGENERATOR"
value="CA0B0334" />
      <input type="hidden" name="__EVENTVALIDATION" id="__EVENTVALIDATION"
value="sHulxT9NcEi+GGq0r+Er95BjQiBRRhzKxjakcSiPuY0FvmpALb5Fz09PZAvovYxyPqzevTG8
+n8GZFNUxx7U57gBDj+I6QG05EPrXE65ew8x5tsWSBxc3CvQlCM+BcFJX7l4WYQwlWt2PmD6wgWD0tvFls5HivIFevyb
t7vv0GVhAkWSbN2/0git+B18U32PLUFBpkX+RJNrD0TbZNHGwk3sJtIBETRmi4naI9WAvY75PAbbOmFtxppvmegiZlKD
way9unOyoR29Vr85hhI8oeX4XA1PbVVV/mpqm0lc23vTGb5fYvfmnoBIn5e2Ud+ARiz19JEzahLtDPmqZdgtcOA5dwab
mrMs0DpeOStk0rgrNBmLxc+q51cDOoD6sctT" />
</div></form>
```

These fields should be excluded when serializing the form fields. This can be done using the `.not()` method, as follows:

```
var strData =
$("#form1").find("input,select,textarea").not("#__VIEWSTATE
").not("#__VIEWSTATEGENERATOR").not("#__EVENTVALIDATION").
serializeArray();
```

3. The next step is to build the Ajax request that is to be sent to the page method. To do this, first, get the URL of the current page:

```
var loc = window.location.href;
```

Send an Ajax request to the page method with the required options:

```
$.ajax({
  type: "POST",
  url: loc + "/ProcessForm",
  data: JSON.stringify({ formData: strData }),
  dataType: "json",
  contentType: "application/json;charset=utf-8",
  timeout: 5000,
  cache: false,
  success: function (response) {
    $("#<%=pnlResult.ClientID%>").html(response.d).show();
  },
  error: function (jqXHR, textStatus, errorThrown) {
    if (textStatus == "error") {
      alert("An error has occurred: " + jqXHR.status + " "
+ jqXHR.statusText);
    }
  }
});
```

In the preceding Ajax call, the following options are set:

- The `type` of request is set to `POST`.
- The `url` of the request is set to the address of the page method.
- The form `data` is sent in a JSON format using `JSON.stringify()`.
- The `type` of the expected response is set to `json`.
- The content type of the sent request is set to `application/json` and the character set to `utf-8`.
- A `timeout` of `5000` milliseconds is defined after which the request will be terminated if the server fails to respond.
- The cache is set to `false` so that the response is not cached.
- A callback function is defined when the response is successful. In this function, the response is displayed in the Panel control and the control is made visible:

  ```
  $("#<%=pnlResult.ClientID%>").html(response.d).show();
  ```

  [Note that the Ajax response is contained in `response.d`.]

- A callback function is defined when the response is unsuccessful. If the returned value of `textStatus` is `error`, then the `status` and `statusText` parameter of the `XmlHttpRequest` object is displayed.

4. The **Reset** button has an event handler attached to it, as follows:

   ```
   $("#<%=btnReset.ClientID%>").on("click", function (evt)
   {…});
   ```

   Inside the preceding handler, the page is prevented from posting back, as follows:

   ```
   evt.preventDefault();
   ```

   Next, all the fields are reset one after the other:

   - First, the `TextBox` controls are cleared:

     ```
     $("#<%=txtName.ClientID%>").val("");
     $("#<%=txtEmail.ClientID%>").val("");
     ```

   - The `DropdownList` control is set to the default value:

     ```
     $("#<%=ddlGender.ClientID%>").val("");
     ```

   - If any radio button is selected, it is cleared:

     ```
     $("#<%=rblType.ClientID%> :checked").prop("checked",
     false);
     ```

- All checkboxes are also cleared:

```
$("#<%=chkContact.ClientID%> :checked").prop("checked",
false);
```

See also

The *Uploading files in MVC* recipe

Uploading files in MVC

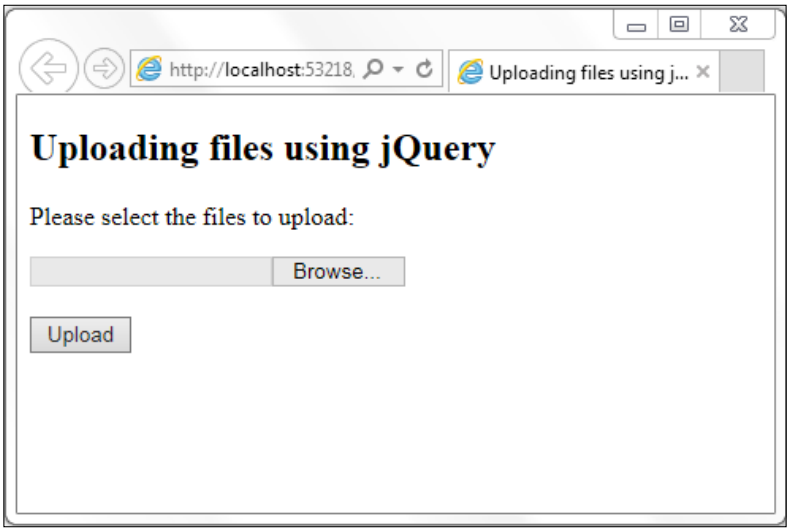
In this example, we will create an MVC website and use jQuery to upload files on the server using HTTP `POST`. The constructs used in this example are summarized as follows:

Construct	Type	Description
`$("#identifier")`	jQuery selector	This selects an element based on its ID.
`$.ajax()`	jQuery function	This posts an Ajax request to the server with the set options.
`click`	jQuery event	This is fired when you click on an element. It corresponds to the JavaScript `click` event.
`event.preventDefault()`	jQuery method	This prevents the default action of the event from being triggered.
`FormData()`	Web API	This creates a new `FormData` object.
`FormData.append()`	Web API method	This adds a key/value pair to the `FormData` object.
`.files`	HTML5 property	This returns a `FileList` object consisting of selected files.
`.get(0)`	jQuery method	This returns the first element from the jQuery array.
`.length`	jQuery property	This gets the number of elements in the jQuery object.
`.on()`	jQuery event binder	This attaches an event handler for one or more events to the matched elements.
`.val()`	jQuery method	This returns the value of the first matched element or sets the value of every matched element.

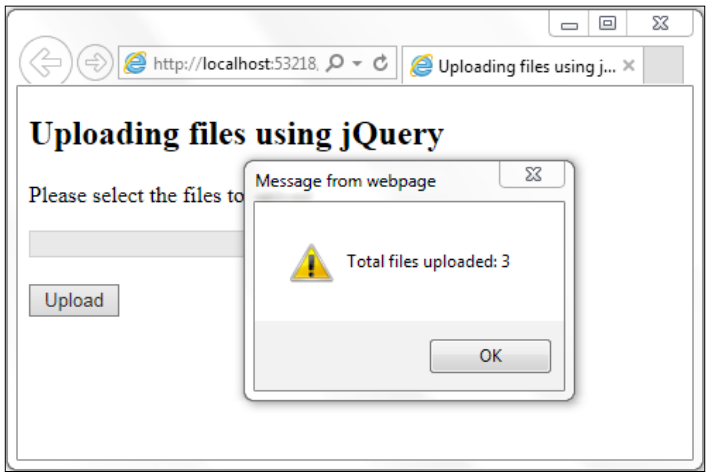
Getting ready

To build a file upload form in MVC, follow these steps:

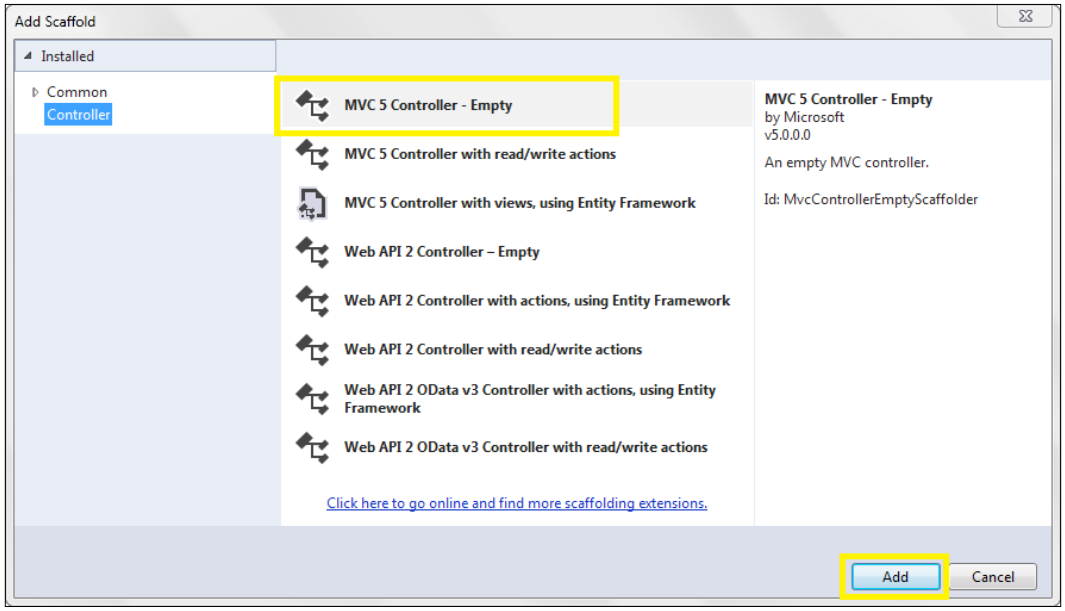
1. In this example, we will create a simple file upload form, as shown in the following screenshot:



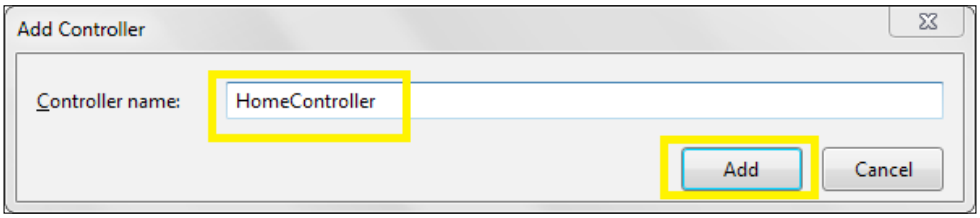
After selecting one or more files and clicking on the **Upload** button, the files are uploaded on the server using Ajax. If the process is successful, a notification is displayed, as shown in the following screenshot:



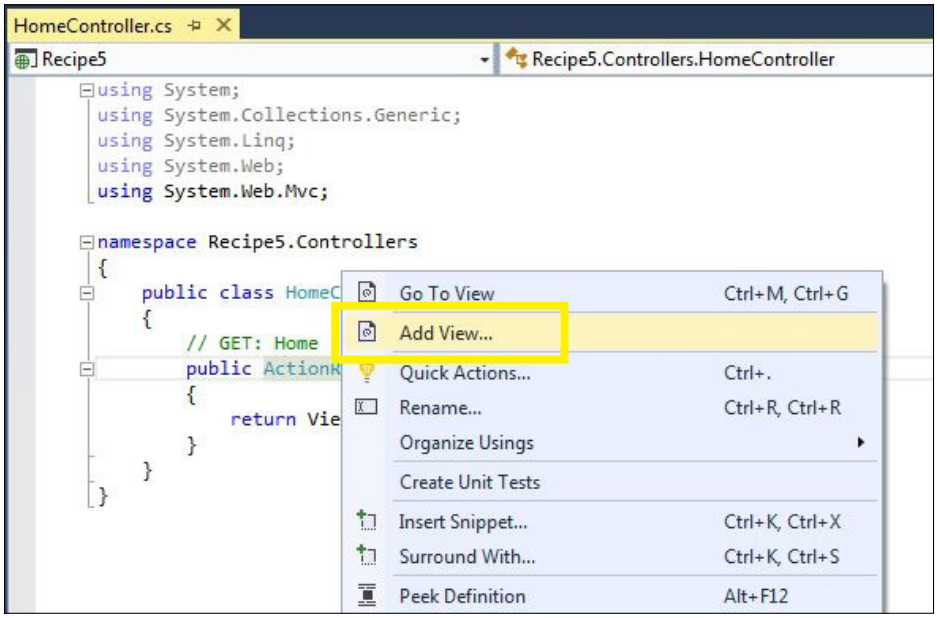
2. Let's get started by launching a new **ASP.NET Web Application** project in Visual Studio. Select the **Empty** template and check the **MVC** box. This will create an empty project with MVC folders.
3. Add a `Scripts` folder to the project and add the jQuery library to this folder.
4. Create a folder called `uploads`. This folder will be used to save the uploaded files.
5. Right-click on the **Controllers** folder in the **Solution Explorer** tab, and go to **Add | Controller**. From the dialog box that is launched, select **MVC 5 Controller – Empty**, and click on **Add**, as shown in the following screenshot:



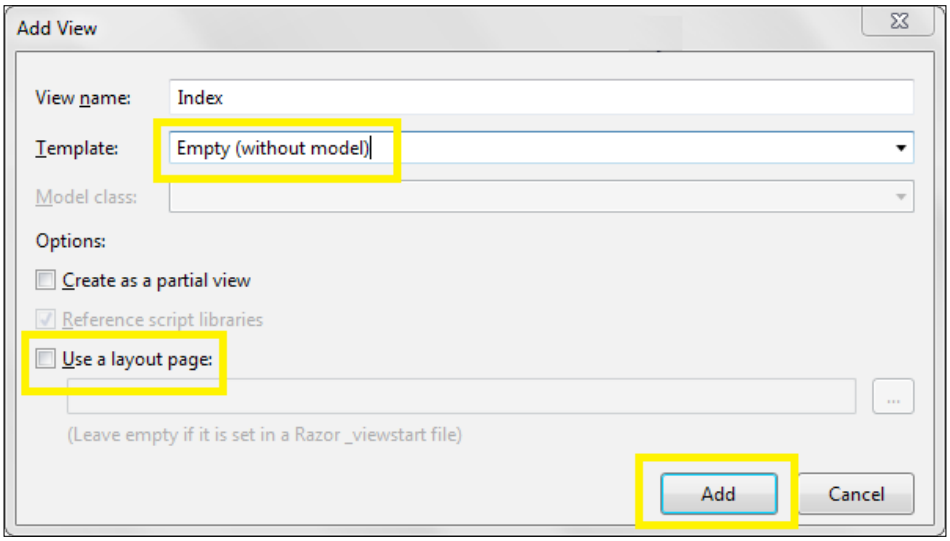
This will display a popup to help you enter the controller name. Type `HomeController` for the name of the controller, and click on **Add**, as shown here:



6. Next, open the `HomeController` file, and right-click on the `Index action` method. This will display a context menu, as shown in the following screenshot. Click on **Add View** in the context menu:



This will display the **Add View** dialog box. Select **Empty (without model)** from the **Template** field. Uncheck the **Use a layout page** option, and click on the **Add** button:



7. Now, open the **View** and include the jQuery library on the page. Add the following markup to create an upload form:

```
<p>Please select the files to upload: </p>
<input id="fileUpload" type="file" multiple/>
<br/><br/>
<input id="btnUpload" type="button" value="Upload" />
```

   Note that the file upload element uses `multiple` to allow the user to select more than one file for upload.

8. Now, let's focus on the controller. In the `HomeController` file, add the `System.IO` namespace at the top of the file.

   For VB, the namespace is as follows:

```
Imports System.IO
```

   For C#, the namespace is as follows:

```
using System.IO;
```

9. Add an action method to the controller to handle HTTP `POST` requests. We will name this method, `UploadFiles()`, and it will be responsible for copying the received files in the uploads folder.

   For VB, the code is as follows:

```
<HttpPost>
Public Function UploadFiles() As String
  Dim strReturn As String = String.Empty
  Dim totalFiles As Integer = Request.Files.Count
  Dim i As Integer
  Try
    For i = 0 To totalFiles - 1
      Dim fileToUpload = Request.Files(i)
      Dim uploadPath =
Path.Combine(Server.MapPath("~/uploads/"),
Path.GetFileName(fileToUpload.FileName))
      fileToUpload.SaveAs(uploadPath)
    Next
  Catch ex As Exception
    strReturn = String.Format("An error has occurred:
{0}", ex.Message.ToString())
  End Try

  If strReturn.Equals(String.Empty) Then
    strReturn = String.Format("Total files uploaded: {0}",
totalFiles)
  End If
```

```
  Return strReturn
End Function
```

For C#, the code is as follows:

```
[HttpPost]
public string UploadFiles()
{
  string strReturn = String.Empty;
  int totalFiles = Request.Files.Count;
  try
  {
    for (int i = 0; i < totalFiles; ++i)
    {
      var fileToUpload = Request.Files[i];
      var uploadPath = Path.Combine(Server.MapPath("~/uploads/"),
Path.GetFileName(fileToUpload.FileName));
      fileToUpload.SaveAs(uploadPath);
    }
  }catch (Exception ex)
  {
    strReturn = String.Format("An error has occurred: {0}",
ex.Message.ToString());
  }
  if (strReturn == String.Empty)
    strReturn = String.Format("Total files uploaded: {0}",
totalFiles);
  return strReturn;
}
```

In the preceding action method, each file in the `Request.Files` object is saved in the `uploads` folder. The file upload snippet is enclosed in a `try…catch` block, and exceptions, if any, are returned to the user in the Ajax response. If the upload is successful, the total number of files is returned in the Ajax response.

How to do it...

Add the following jQuery code to a `script` block in the `Index` view:

```
$(document).ready(function () {
  $("#btnUpload").on("click", function (evt) {
    evt.preventDefault();
    var filesToUpload = $("#fileUpload").get(0).files;
    var fd = new FormData();
    for (var i = 0; i < filesToUpload.length; ++i) {
```

```
        fd.append(filesToUpload[i].name, filesToUpload[i]);
      }
      $.ajax({
        method: "POST",
        url: "/Home/UploadFiles",
        contentType: false,
        data: fd,
        dataType: "json",
        cache: false,
        processData: false,
        error: function (jqXHR, textStatus, errorThrown) {
          if (textStatus == "error") {
            alert("An error has occurred: " + jqXHR.status + " " +
jqXHR.statusText);
          }
        },
        complete: function (response) {
          $("#fileUpload").val("");
          alert(response.responseText);
        }
      });
    });
});
</script>
```

At times, Visual Studio may skip breakpoints during debugging. In such a situation, the debugger statement can be included in the script, as follows:

```
debugger;
```

The preceding statement will create a breakpoint in the script.

How it works...

The uploading of files in MVC works as follows:

1. In the jQuery script, an event handler is attached to the `click` event of the **Upload** button, as follows:

   ```
   $("#btnUpload").on("click", function (evt) {…});
   ```

In the preceding event handler, the page is prevented from posting back using the `preventDefault()` method:

```
evt.preventDefault();
```

2. Next, get the list of files selected by the user in the file input element as a `FileList` object:

```
var filesToUpload = $("#fileUpload").get(0).files;
```

3. Instantiate a `FormData` object. This object consists of a list of key/value pairs. The key refers to the filename while the value refers to the `File` object:

```
var fd = new FormData();
```

4. Loop through each file in the `FileList` object and add a key/value pair to the `FormData` object:

```
for (var i = 0; i < filesToUpload.length; ++i) {
  fd.append(filesToUpload[i].name, filesToUpload[i]);
}
```

5. Next, post an Ajax request to the controller action method:

```
$.ajax({
  method: "POST",
  url: "/Home/UploadFiles",
  contentType: false,
  data: fd,
  dataType: "json",
  cache: false,
  processData: false,
  error: function (jqXHR, textStatus, errorThrown) {
    if (textStatus == "error") {
      alert("An error has occurred: " + jqXHR.status + " "
+ jqXHR.statusText);
    }
  },
  complete: function (response) {
    $("#fileUpload").val("");
    alert(response.responseText);
  }
});
```

In the preceding statement, the following options are set:

- The `type` of request is set to `POST`.
- The `url` is set to the address of the `UploadFiles` action method.

- The `contentType` parameter is set to `false` to prevent jQuery from adding the content type header.
- Send the `FileData` object as the request data.
- Set the `dataType` parameter of the response to `json`.
- Set the `cache` parameter to `false` so that the response will not be cached.
- Set `processData` to `false` so that the file data is not processed prior to sending it to the server.
- Define a callback function for an unsuccessful Ajax request. If `error` is returned as `textStatus`, then the `status` and `statusText` parameter of the `XmlHttpRequest` object is displayed.
- Define a callback function when the request is completed. In this function, clear the file upload input element:

  ```
  $("#fileUpload").val("");
  ```

- Next, display the response returned from the server:

  ```
  alert(response.responseText);
  ```

It is important to set the `contentType` and `processData` options to `false` for the upload process to be successful.

See also

The *Using Ajax to load scripts in web pages* recipe

Exporting the GridView data in the CSV format

In this example, let's export the data of a `GridView` control to the **CSV** (**Comma Separated Values**) format. The same script can also be used to export data from HTML tables or ASP.NET `Table` controls. The constructs used in this example are summarized as follows:

Construct	Type	Description
`$("#identifier")`	jQuery selector	This selects an element based on its ID.
`$("html_tag")`	jQuery selector	This selects all elements with the specified HTML tag.
`$(this)`	jQuery object	This refers to the current jQuery object.

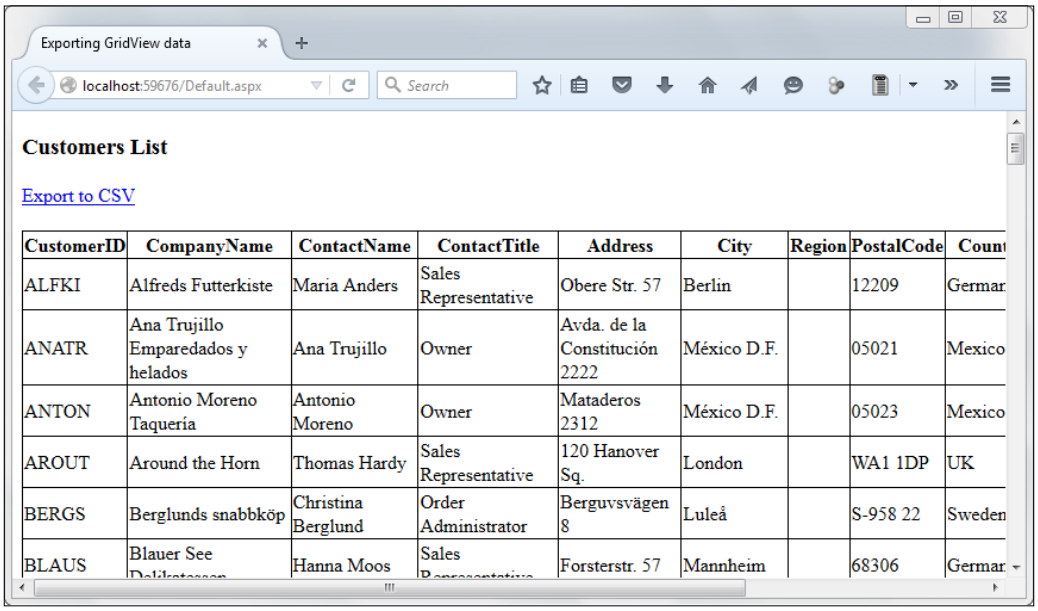
Construct	Type	Description
`.attr("name")` or `.attr("name", "value")`	jQuery method	This returns a string with the required attribute value of a matched element. It can also be used to set the attribute to the required value.
`click`	jQuery event	This is fired when you click on an element. It corresponds to the JavaScript `click` event.
`.each()`	jQuery method	This iterates over the matched elements and executes a function for each element.
`encodeURIComponent()`	JavScript method	This encodes a uniform resource identifier by replacing each instance of certain characters by escape characters.
`:eq(i)`	jQuery filter	This selects an element with an index equal to `i` from the matched elements.
`.find()`	jQuery method	This finds all elements that match the filter.
`.get()`	jQuery method	This gets the DOM elements matched by the jQuery object.
`:gt(i)`	jQuery filter	This selects elements with an index greater than `i` from the matched elements.
`.join()`	JavaScript method	This joins the elements of an array to a comma separated string.
`.map()`	jQuery method	This executes a function on each matched element and returns a new jQuery object with the updated values.
`.on()`	jQuery event binder	This attaches an event handler for one or more events to the matched elements.
`.replace()`	JavaScript method	This searches for the specified substring in a string and replaces it with another substring.

Construct	Type	Description
`.text()`	jQuery method	This returns the combined text content of each of the matched elements or sets the text content of every matched element.
`.trim()`	JavaScript method	This removes leading and trailing whitespaces from the string.

Getting ready

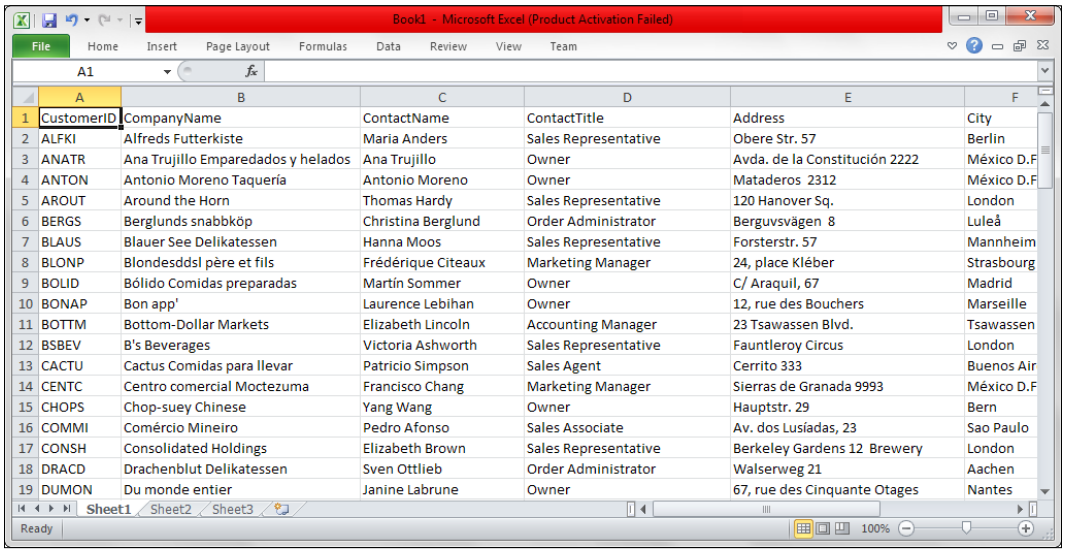
Follow these steps to setup a GridView control on a page:

1. In this example, let's create a web page to display customer records from the **Northwind** database, as shown in the following screenshot:



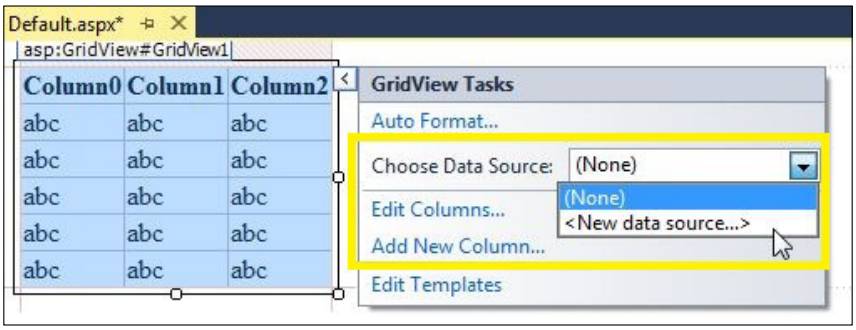
CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode	Count
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57	Berlin		12209	Germar
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la Constitución 2222	México D.F.		05021	Mexico
ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner	Mataderos 2312	México D.F.		05023	Mexico
AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.	London		WA1 1DP	UK
BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator	Berguvsvägen 8	Luleå		S-958 22	Sweden
BLAUS	Blauer See	Hanna Moos	Sales	Forsterstr. 57	Mannheim		68306	Germar

After clicking on the **Export to CSV** link, the data can be downloaded in CSV, as shown in the following screenshot:



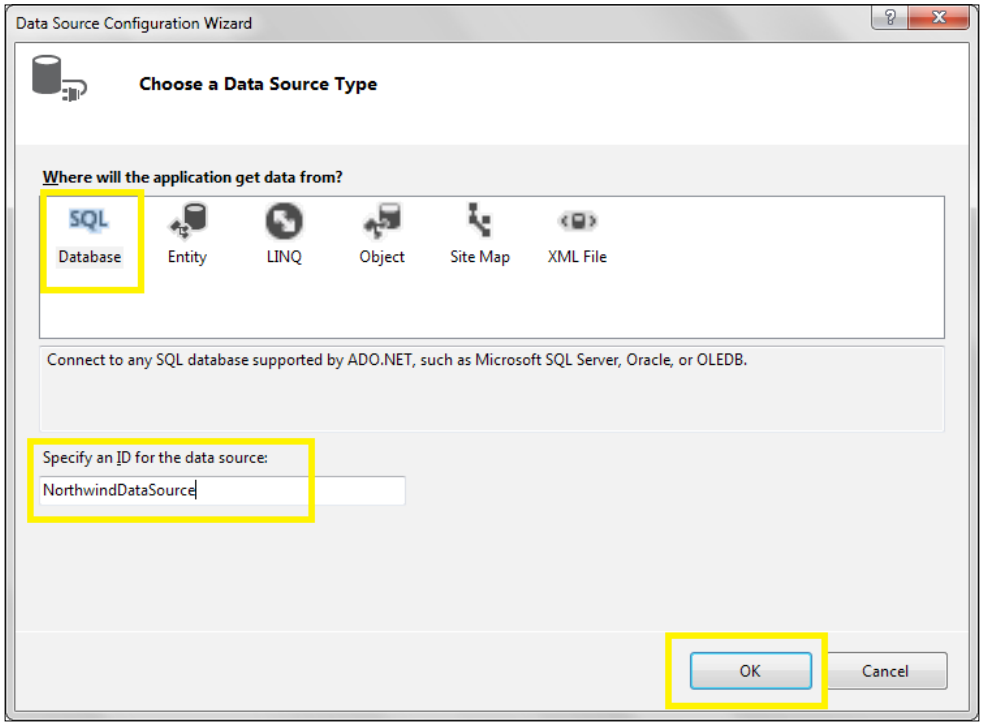
CustomerID	CompanyName	ContactName	ContactTitle	Address	City
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57	Berlin
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la Constitución 2222	México D.F
ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner	Mataderos 2312	México D.F
AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.	London
BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator	Berguvsvägen 8	Luleå
BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative	Forsterstr. 57	Mannheim
BLONP	Blondesddsl père et fils	Frédérique Citeaux	Marketing Manager	24, place Kléber	Strasbourg
BOLID	Bólido Comidas preparadas	Martín Sommer	Owner	C/ Araquil, 67	Madrid
BONAP	Bon app'	Laurence Lebihan	Owner	12, rue des Bouchers	Marseille
BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln	Accounting Manager	23 Tsawassen Blvd.	Tsawassen
BSBEV	B's Beverages	Victoria Ashworth	Sales Representative	Fauntleroy Circus	London
CACTU	Cactus Comidas para llevar	Patricio Simpson	Sales Agent	Cerrito 333	Buenos Air
CENTC	Centro comercial Moctezuma	Francisco Chang	Marketing Manager	Sierras de Granada 9993	México D.F
CHOPS	Chop-suey Chinese	Yang Wang	Owner	Hauptstr. 29	Bern
COMMI	Comércio Mineiro	Pedro Afonso	Sales Associate	Av. dos Lusíadas, 23	Sao Paulo
CONSH	Consolidated Holdings	Elizabeth Brown	Sales Representative	Berkeley Gardens 12 Brewery	London
DRACD	Drachenblut Delikatessen	Sven Ottlieb	Order Administrator	Walserweg 21	Aachen
DUMON	Du monde entier	Janine Labrune	Owner	67, rue des Cinquante Otages	Nantes

2. To get started, launch a new **ASP.NET Web Application** in Visual Studio using the **Empty** template and name it `Recipe6` (or any other suitable name).
3. Create a `Scripts` folder in the project and include the jQuery library in this folder.
4. Add a web form named `Default.aspx`. Include the jQuery library on the form.
5. Go to **Toolbox** | **Data**, and drag and drop a `GridView` control on the form.
6. In the **Design** mode, mouse over the `GridView` control until a small arrow icon appears in the top-right corner. Click on the arrow icon to display the **GridView Tasks** menu, as shown in the following screenshot:

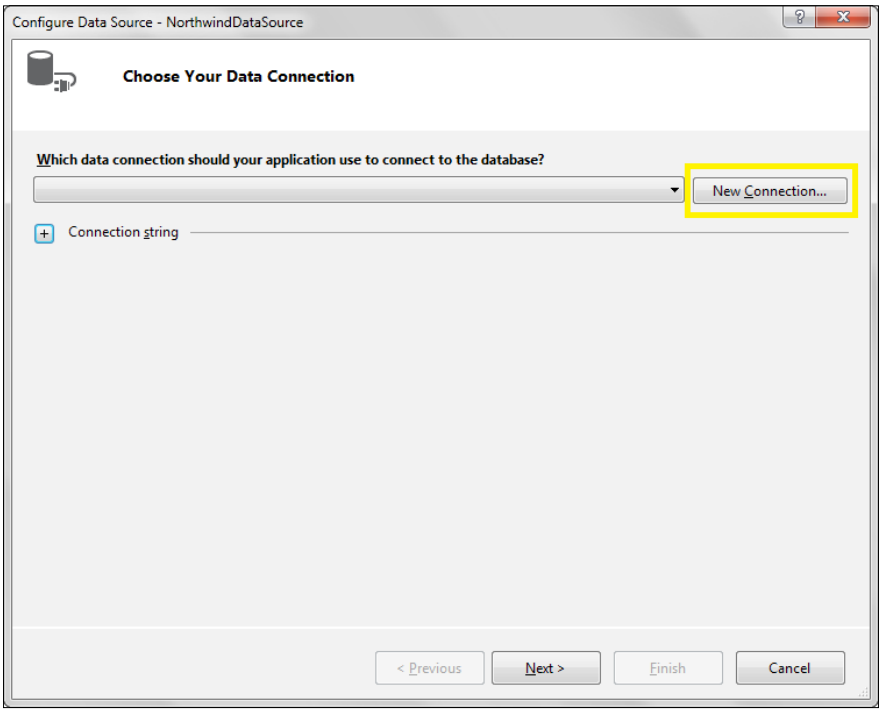


From the **Choose Data Source** dropdown, select **<New data source...>**.

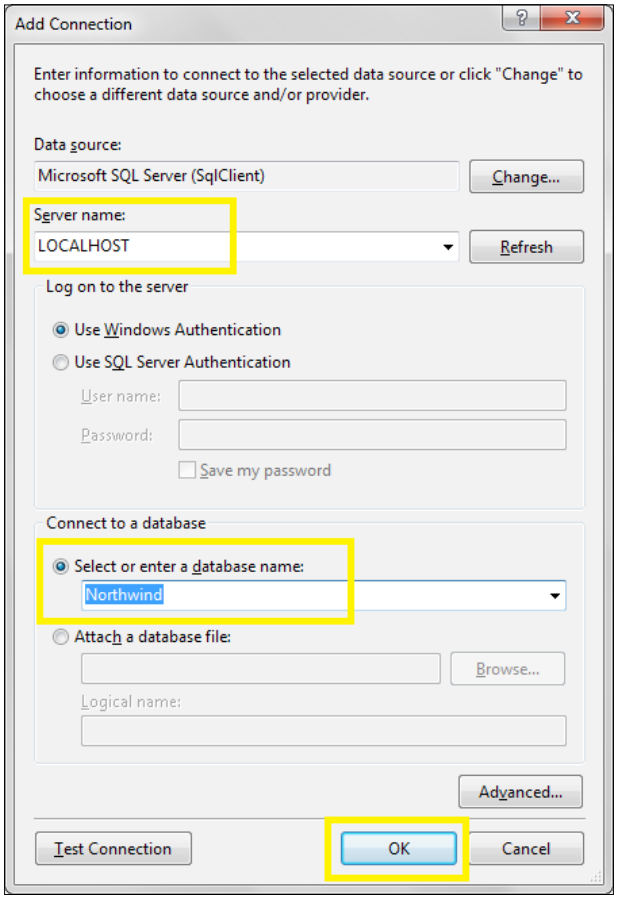
7. From the following dialog box of the **Data Source Configuration** wizard, select **SQL Database**. Type `NorthwindDataSource` for the ID of the data source, and click on the **OK** button:



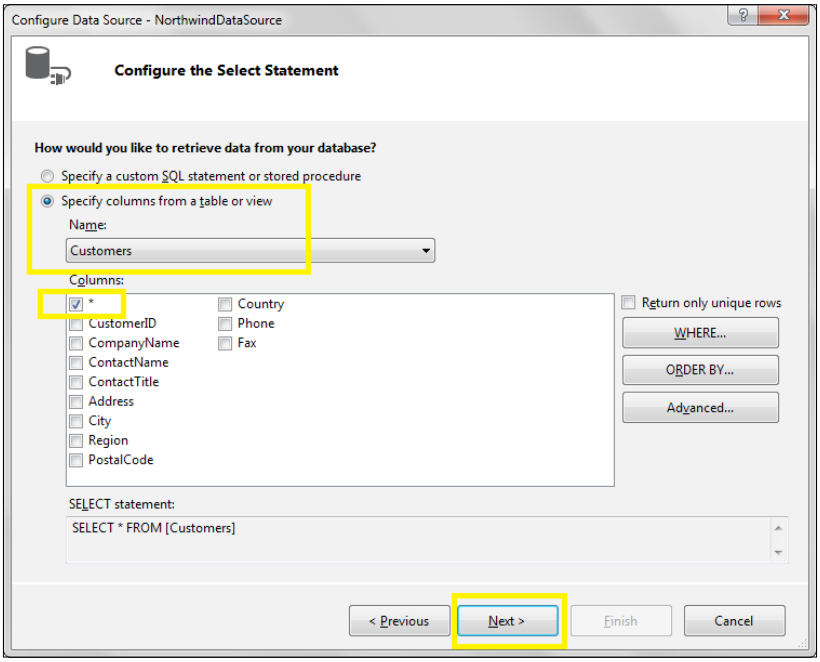
8. Click on the **New Connection** button:



This will launch the **Add Connection** dialog box, as shown in the following screenshot. Enter `LOCALHOST` for the server name, select **Northwind** as the database name, and click on **OK**:



9. The following screenshot displays the configuration of the **SELECT statement**. Choose the `Customers` table and select all the columns. Finish the configuration wizard:



Note that we are using Windows Authentication for all database driven examples in this book. Hence in the MS SQL Server, it is important to give permission to the Windows account to access the Northwind database.

10. Now that the database setup is completed on the web form, add a `LinkButton` control above the `GridView` control to get the following markup on the page:

```
<asp:LinkButton ID="btnExport" runat="server">Export to
CSV</asp:LinkButton>
<br /><br />
<asp:GridView ID="customersGridView" runat="server"
AutoGenerateColumns="False" DataKeyNames="CustomerID"
DataSourceID="NorthwindDataSource">
  <Columns>
    <asp:BoundField DataField="CustomerID" HeaderText="CustomerID"
ReadOnly="True" SortExpression="CustomerID" />
    <asp:BoundField DataField="CompanyName"
HeaderText="CompanyName" SortExpression="CompanyName" />
```

```
        <asp:BoundField DataField="ContactName"
    HeaderText="ContactName" SortExpression="ContactName" />
        <asp:BoundField DataField="ContactTitle"
    HeaderText="ContactTitle" SortExpression="ContactTitle" />
        <asp:BoundField DataField="Address" HeaderText="Address"
    SortExpression="Address" />
        <asp:BoundField DataField="City" HeaderText="City"
    SortExpression="City" />
        <asp:BoundField DataField="Region" HeaderText="Region"
    SortExpression="Region" />
        <asp:BoundField DataField="PostalCode" HeaderText="PostalCode"
    SortExpression="PostalCode" />
        <asp:BoundField DataField="Country" HeaderText="Country"
    SortExpression="Country" />
        <asp:BoundField DataField="Phone" HeaderText="Phone"
    SortExpression="Phone" />
        <asp:BoundField DataField="Fax" HeaderText="Fax"
    SortExpression="Fax" />
      </Columns>
    </asp:GridView>
    <asp:SqlDataSource ID="NorthwindDataSource" runat="server"
    ConnectionString="<%$
    ConnectionStrings:NorthwindConnectionString %>"
    SelectCommand="SELECT * FROM
    [Customers]"></asp:SqlDataSource>
```

How to do it...

Add the following jQuery code to a `script` block on the page:

```
<script type="text/javascript">
  $(document).ready(function() {
    $("#<%=btnExport.ClientID%>").on("click", function() {
      var csvContent = "";
      var fileName = "export.csv";
      var newline = "\r\n";

      //Write the header row
      var $header =
$("#<%=customersGridView.ClientID%>").find("tr:eq(0)");
      var $headercols = $header.find("th");
      var csv = $headercols.map(function(j, col) {
        return '"' + $(col).text().replace('/"', '/"/"') + '"';
      }).get().join();
      csvContent = csv + newline;
```

```
      //Write all the content rows
      var $rows =
$("#<%=customersGridView.ClientID%>").find("tr:gt(0)");
      $rows.each(function(i, row) {
        var $cols = $(row).find("td");
        var csv = $cols.map(function(j, col) {
          if ($(col).text().trim() != "")
            return '"' + $(col).text().replace('"', '""') + '"';
          else
            return '""';
        }).get().join();
        csvContent += csv + newline;
      });
      csvContent = "data:application/csv;charset=utf-8," +
encodeURIComponent(csvContent);
      $(this).attr({
        download: fileName,
        href: csvContent,
        target: "_blank"
      });
    });
  });
</script>
```

How it works...

The export of GridView data to CSV format works as follows:

1. In the jQuery script, an event handler is attached to the `click` event of the `LinkButton` control:

   ```
   $("#<%=btnExport.ClientID%>").on("click", function () {…});
   ```

   Note that we will not use `event.preventDefault()` in this handler since we want the default action of the link to be executed.

2. Initialize a variable named `csvContent`. This will be used to build a comma separated string from the contents of the `GridView` control:

   ```
   var csvContent = "";
   ```

3. Initialize the filename to export the data as required:

   ```
   var fileName = "export.csv";
   ```

4. Define a variable named `newline` to hold the carriage return and newline characters:

   ```
   var newline = "\r\n";
   ```

5. Next, build the header row. To do this, first, find the header row, that is, the `tr` element at index `0`, as follows:

```
var $header = $("#<%=customersGridView.ClientID%>").
find("tr:eq(0)");
```

Then, determine the header columns by filtering the preceding row using the `th` elements:

```
var $headercols = $header.find("th");
```

Build a comma separated string by mapping the header columns to return the column text enclosed in double quotes. If the column content has double quotes, it is replaced by two double quotes:

```
var csv = $headercols.map(function (j, col) {
  return '"' + $(col).text().replace('/"', '/"/"') + '"';
}).get().join();
```

The `.map()` method returns a jQuery object containing an array. We use `.get()` to retrieve the array, and the corresponding elements are joined to a comma separated string.

> The `GridView` columns may have commas in the content. To avoid the CSV file from incorrectly splitting up at the commas in the content, all column values are enclosed in double quotes. Double quotes in the column values, if any, are escaped by replacing them with two double quotes.

6. Next, add the newline to the preceding `csv` string:

```
csvContent = csv + newline;
```

7. After building the header row, we use a similar process on each row of the `GridView` control:

```
$rows.each(function (i,row) {…});
```

In the preceding function, select the columns in each row:

```
var $cols = $(row).find("td");
```

Use the `.map()` method and return a comma separated string that consists of the column values enclosed in double quotes. If the column content is empty, an empty string enclosed in double quotes is returned for that column:

```
var csv = $cols.map(function (j, col) {
  if ($(col).text().trim() != "")
    return '"' + $(col).text().replace('"', '""') + '"';
  else
    return '""';
}).get().join();
```

Append the comma separated row content to the `csvContent` string:

```
csvContent += csv + newline;
```

8. Attach a header to the content to indicate the data as CSV and the character set as `utf-8`:

```
csvContent = "data:application/csv;charset=utf-8," +
encodeURIComponent(csvContent);
```

9. Lastly, update the attribute of the `LinkButton` control to attach the CSV content generated from the `GridView` control as a downloadable file, as shown in the following code:

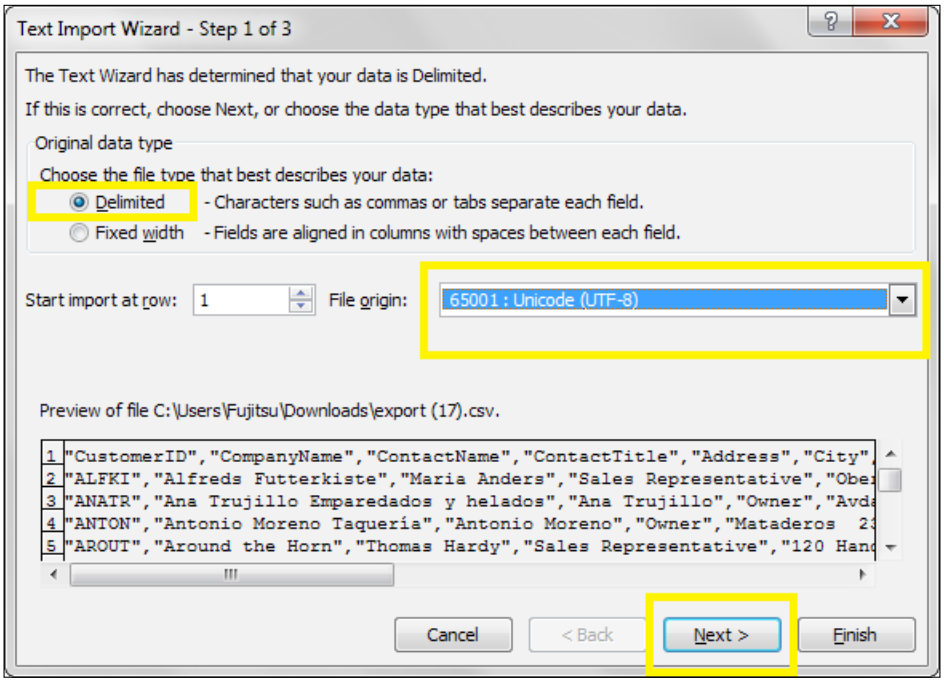
```
$(this).attr({
  download: fileName,
  href: csvContent,
  target: "_blank"
});
```

Note that there is a limitation in using this script with IE browsers. In IE, the CSV content will not be attached as a downloadable file, and step 9 will not work. To overcome this issue, we can provide a multiline textbox control on the form, and add the CSV content to this control.

There's more...

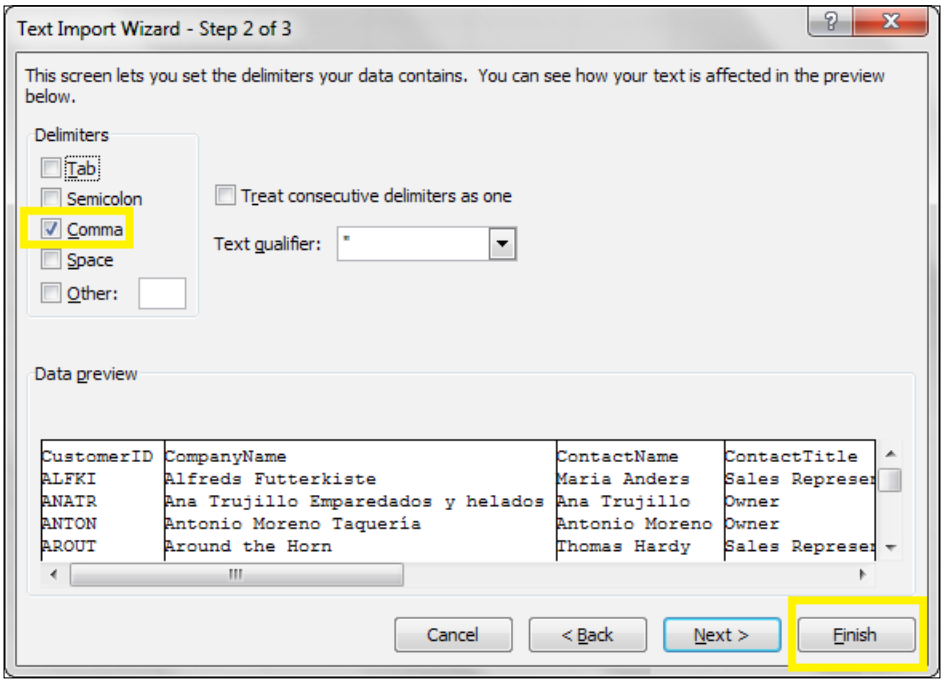
If the `GridView` control has accented (non-English) characters, the exported CSV file may not display them correctly. To see the data correctly, perform the following steps:

1. Launch Microsoft Excel.
2. From the **File** menu, go to **Data** | **Get External Data** | **From Text**. This will launch a browse window. Select the CSV file exported in the previous section, and click on **Open**.
3. Next, **Text Import Wizard** will be launched, as shown in the following screenshot:



Select **Delimited** as the file type, and choose **65001: Unicode (UTF- 8)** as the encoding. Click on the **Next** button.

4. In the next step, select **Comma** as the file delimiter. Click on **Finish** to complete the wizard:



This will display the accented characters correctly in the CSV file.

See also

The *Infinite scrolling* recipe