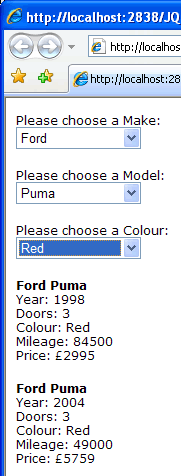
Cascading DropDownLists with jQuery and ASP.NET

The sample page for this exercise will present the user with a DropDownLis containing a selection of Makes of car. Once they have made their selection, a second DropDown will populate with available Models for the chosen Make. Selecting a Model will populate the avaialable Colours for that Model, and car details for all cars that match on Make, Model and Colour will be presented to the user when they have selected the Colour they want.



For this to happen, the **WebService** create a Web Service first. But in the WebService update the <http://tempuri.org/> to the correct one, in this case is

[WebService(Namespace = "http://localhost:2523/WebSite3/")]

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Services;

using System.Web.Script.Services;

public class Car

{

public string Make;

public string Model;

public int Year;

public int Doors;

public string Colour;

public float Price;

public int Mileage;

}

[WebService(Namespace = "http://tempuri.org/")]

[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1\_1)]

**[ScriptService]**

public class **CarService** : WebService

{

List<Car2> Cars = new List<Car2>{

new Car{Make="Ford",Model="Focus",Year=2002,Doors=5,Colour="Black",Price=3250f,Mileage=68500},

new Car{Make="Toyota",Model="Previa",Year=1998,Doors=5,Colour="Green",Price=2695f,Mileage=72400},

new Car{Make="Toyota",Model="Corolla T3",Year=2004,Doors=5,Colour="Blue",Price=5995f,Mileage=71000},

new Car{Make="Toyota",Model="Yaris",Year=2005,Doors=3,Colour="Grey",Price=5350f,Mileage=39000},

new Car{Make="Ford",Model="Fiesta",Year=2004,Doors=3,Colour="Red",Price=5759f,Mileage=49000},

new Car{Make="Ford",Model="Mondeo",Year=2007,Doors=5,Colour="Gold",Price=12250f,Mileage=8500},

new Car{Make="Toyota",Model="Verso",Year=2008,Doors=5,Colour="White",Price=12995f,Mileage=5800},

new Car{Make="Ford",Model="Mondeo",Year=2004,Doors=5,Colour="Green",Price=8759f,Mileage=66000},

new Car{Make="Ford",Model="Focus",Year=2007,Doors=5,Colour="Grey",Price=9950f,Mileage=19000},

new Car{Make="Ford",Model="Fiesta",Year=2004,Doors=3,Colour="Grey",Price=5759f,Mileage=49000}

};

    [WebMethod]

    public List<string> GetCarMakes()

    {

      var query = (from c in Cars

                  orderby c.Make

                  select c.Make).Distinct();

      return query.ToList();

    }

    [WebMethod]

    public List<string> GetModels(string make)

    {

      var query = (from c in Cars

                  where c.Make == make

                  orderby c.Model

                  select c.Model).Distinct();

      return query.ToList();

    }

    [WebMethod]

    public List<string> GetCarsByColour(string make, string model)

    {

      var query = (from c in Cars

                  where c.Make == make && c.Model == model

                  orderby c.Colour

                  select c.Colour).Distinct();

      return query.ToList();

    }

    [WebMethod]

    public List<Car> GetCarListByColour(string make, string model, string colour)

    {

      var query = from c in Cars

                  where (c.Make == make &&

                      c.Model == model &&

                        c.Colour == colour)

                    select c;

      return query.ToList();

    }

}

**Without the [ScriptService] [System.Web.Script.Services.ScriptService] The ajax will not possible access the web service.**

Three of the additional methods return types of List<string>. These are used to populate the DropDownLists. The final method retrieves a List<Car> that meets the Model, Make and Colour criteria that have been selected using the DropDownLists.

I say "DropDownLists", but actually there is only one in the aspx file. The other two are straightforward HTML <select> elements. It's generally good practice to only use ASP.NET Server-Side controls where they are actually needed. Since all the work in populating and referencing the second and third list is done on the client, I don't need to reference them from Code-Behind. Consequently, there is no need to have runat="server" controls.

<form id="form1" runat="server">

<div>

 <div>

   <p>

    <label>

      Please choose a Make:</label><br />

    <asp:DropDownList ID="ddlMake" runat="server" />

   </p>

 </div>

 <div>

  <p>

    <label>

      Please choose a Model:</label><br />

    <select id="ddlModel">

    </select></p>

 </div>

 <div>

  <p>

    <label>

      Please choose a Colour:</label><br />

    <select id="ddlColour">

    </select></p>

 </div>

 <div id="output"></div>

</div>

</form>

The server control DropDownList will be filled from code-behind on Page\_Load:

protected void Page\_Load(object sender, EventArgs e)

{

  if (!IsPostBack)

  {

**CarService service = new CarService();**

**List<string> Makes = service.GetCarMakes();**

    ddlMake.DataSource = Makes;

    ddlMake.DataBind();

    ddlMake.Items.Insert(0, " -- Select Make -- ");

  }

}

AJAX is what this example is all about, so let's get into the client-side code:

<title></title>

<script src="Scripts/jquery-1.2.6.min.js" type="text/javascript"></script>

<script src="Scripts/jquery.selectboxes.min.js" type="text/javascript"></script>

<script type="text/javascript">

$(function () {

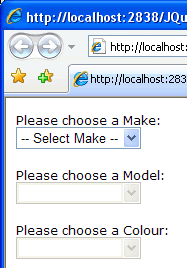
$('#ddlMake').change(getModels);

$('#ddlModel').attr('disabled', true);

$('#ddlColour').attr('disabled', true);

});

Two external script files are referenced: the first is the condensed version of jQuery, while the second is the plugin for managing dropdowns or selects. This is followed by some code that will run in the onload event of the page. It applies an event handler to the onchange event of the "Make" list which calls the getModels() function, and it also sets the second and third selects to disabled.



Choosing an option in the first list will fire the **GetModels** function which is as follows:

function getModels() {

var Fabricante = $('#ddlMake').val();

var ff = document.getElementById('ddlMake').value;

$.ajax({

type: "POST",

url: "**CarService.asmx/GetModels**",

data: "{make: '" + Fabricante + "'}" ,

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

dataType: "json",

async: true,

success: function (response) {

//alert(respose.d);

var models = (typeof response.d) == 'string' ? eval('(' + response.d + ')') : response.d;

$('#ddlModel').attr('disabled', false).change(getColours).removeOption(/./).addOption('', ' -- Select Model -- ');

$('#ddlColour').attr('disabled', true).removeOption(/./);

for (var i = 0; i < models.length; i++) {

var val = models[i];

var text = models[i];

$('#ddlModel').addOption(val, text, false);

}

}

});

}

It calls a WebMethod called **GetCarsByModel**, with the Make passed as a parameter. Once the response has been received and validated, a number of things happen. The Model select list is enabled, and jQuery chaining is used to first disable the Colour dropdown (in case it was enabled by a previous Make selection, and then add an onchange event handler which fires the getColours() function. Next the dropdown is cleared using the SelectBoxes plugin's removeOption() function, and finally a default option is added to the dropdownlist. All of this on one line.

The Colour dropdown is then treated to chained commands that disable, then clear it of any previous options. Finally the response is iterated and added as option elements to the Model dropdown. As opposed to the response delivered by the WebMethods in the previous article, this response is an object called d whose single property simply contains an array of strings.

**For example, if Ford is selected from the Make dropdown, the response is as follows:**

{"d":["Fiesta","Focus","Ka","Mondeo","Puma"]}

Selecting one of these models will cause the onchange event handler to fire, which calls the getColours() function:

function getColours() {

$.ajax({

type: "POST",

url: "CarService.asmx/GetCarsByColour",

data: "{make: '" + $('#ddlMake').val() + "', model: '" + $('#ddlModel').val() + "'}",

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

dataType: "json",

success: function (response) {

var colours = (typeof response.d) == 'string' ? eval('(' + response.d + ')') : response.d;

$('#ddlColour').attr('disabled', false).change(getCarListByColour).removeOption(/./).addOption('', ' -- Select Colour -- ');

for (var i = 0; i < colours.length; i++) {

var val = colours[i];

var text = colours[i];

$('#ddlColour').addOption(val, text, false);

}

}

});

}

Two parameters are passed into the call to the WebMethod in this function, these being the Make dropdown selected value and the Model dropdown selected value. Aside from that the rest of the code should be familiar form the previous function. Notice again the jQuery chaining being employed to enable the Colour dropdown, add an onchange event handler, clear it and finally set a default option. The onchange event handler calls the final function, which results in the collection of cars being retrieved from the Web Service and then displayed in the final div on the page:

function getCarListByColour() {

  $.ajax({

    type: "POST",

   url: "CarService.asmx/GetCarListByColour",

    data: "{make: '" + $('#ddlMake').val() + "', " +

          "model: '" + $('#ddlModel').val() + "', " +

          "colour: '" + $('#ddlColour').val() + "'}",

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

    dataType: "json",

    success: function(response) {

      var cars = (typeof response.d) == 'string' ? eval('(' + response.d + ')') : response.d;

      $('#output').empty();

      for (var i = 0; i < cars.length; i++) {

        $('#output').append('<p><strong>' + cars[i].Make + ' ' +

                              cars[i].Model + '</strong><br /> Year: ' +

                              cars[i].Year + '<br />Price: ' +

                              cars[i].Price + '</p>');

      }

    }

  });

}

</script>

The script simply calls the GetCarListByColour() WebMethod and returns an array of objects as a JSON string. The Cars that are returned are those that match the Make, Model and Colours selected in the DropDowns and the result is written to the div at the bottom of the page.