**Dynamically Populate a Select/Drop Down List using ASP.NET MVC, JQuery and Json**

It has been a while since my last blog due to me being flat out at work during the week and snowboarding on the weekends!

Today the weather is terrible up in the alps and I managed to catch the flu so it was a good time to catch up on some personal projects. I was working on an ASP.NET MVC website for my sister and decided to use AJAX to populate a select list based on a selection in another select list. The ASP.NET MVC framework and JQuery makes it very easy to use AJAX to do this.

This is nothing new but very powerful if you want to create a nice fluent forms user interface. For the most part the process was very easy. However I ran into an issue where JQuery was treating the Json returned from the server as a string instead of an object. Thus making if difficult to retrieve the data I wanted. I will explain how I got around this later in the post.

The first thing you need to do create the server side method. This is very easy using ASP.NET MVC! Of course you would probably get your data from a database of some sort, however this is just a nice way of showing how its done. The method below gets a list of Foo objects based on the FooId parameter using LINQ’s Where method.

public class Foo

{

public int FooId { get; set; }

public string Name { get; set; }

}

[AcceptVerbs(HttpVerbs.Post)]

public ActionResult GetFooList(int FooId)

{

var fooList = new List();

fooList.Add(new Foo { FooId = 1, Name = "One" } );

fooList.Add(new Foo { FooId = 1, Name = "Two" } );

fooList.Add(new Foo { FooId = 2, Name = "Three" });

fooList.Add(new Foo { FooId = 2, Name = "Four" });

return Json(fooList.Where(foo => foo.FooId == FooId).ToList());

}

Now for the client side code. On my page I have two select lists. One is a list of Id’s and the other is a list of names. When a user selects an id from the first select list the second is populated with the names relating to that id. The lists can be created easily using using the ASP.NET MVC Helpers or writing the HTML yourself.



Finally the JQuery code in order to add the functionality to the page. JQuery makes it easy to execute server side methods and receive Json. The code below binds the AJAX post method to the change event of the IdList select list. The post function has three variables; The sever method to execute, the parameters to pass as Json and the function to execute with the data returned from the server. You can either execute a named or a anonymous function. I chose the latter because I am not doing too much inside the function.

$(function() {

$("#IdList").change(function() {

$.getJSON("/Controller/GetFooList", { FooId: $("#IdList").val() },

function(fooList) {

$("#NameList").empty();

$.each(fooList, function(i, foo) {

$("#NameList").append(""+ foo.Name+ "");

});

});

});

});

The biggest issue I came across was, originally, the FooList Json was being represented as a string therefore the JQuery each method was iterating through each character of the string rather then each object. After a bit reading I came across the JSON.parse method in the JQuery library. This method parses the string to Json which means that the JQuery each method functions as I expected.

There we have it. Like I said nothing new here but interesting never the less.

The only thing I don’t like about this code is the append method in my JQuery. Surly there is a better way of adding options to a select element?

**UPDATE 120/01/2011:  I have recently realised that using eval is bad! This is because it will evaluate ANY javascript returned from the server. Therefore there is a chance that someone with malicious intent could insert their own javascript into your request. So instead you should use jQuery’s getJSON method instead. The above code has been modified to use this function.**