Using Auto-Postback with CascadingDropDown

Christian Wenz

Overview

The CascadingDropDown control in the AJAX Control Toolkit extends a DropDownList control so that changes in one DropDownList loads associated values in another DropDownList. (For instance, one list provides a list of US states, and the next list is then filled with major cities in that state.) However when using the CascadingDropDown control, ASP.NET's DropDownList control's AutoPostBack feature does not work, since asynchronously loading data into the list generates an (unnecessary) postback itself. With some JavaScript code, this effect can be avoided.

Steps

In order to activate the functionality of ASP.NET AJAX and the Control Toolkit, the **ScriptManager** control must be put anywhere on the page (but within the **<form>** element):

```
<asp:ScriptManager ID="asm" runat="server" />
```

Then, a DropDownList control is required:

```
<div>
  Vendor: <asp:DropDownList ID="VendorsList" runat="server" />
</div>
```

For this list, a CascadingDropDown extender is added, providing web service URL and method information:

```
<ajaxToolkit:CascadingDropDown ID="ccd1" runat="server"
  ServicePath="CascadingDropdown3.cs.asmx"
   ServiceMethod="GetVendors"
  TargetControlID="VendorsList" Category="Vendor" />
```

The CascadingDropDown extender then asynchronously calls a web service with the following method signature:

```
public CascadingDropDownNameValue[] MethodNameHere(string
   knownCategoryValues, string category)
```

The method returns an array of type CascadingDropDown value. The type's constructor expects first the list entry's caption and then the value (HTML **value** attribute).

```
<%@ WebService Language="C#" Class="CascadingDropdown3" %>

using System.Web.Script.Services;
using AjaxControlToolkit;
```

```
using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Collections.Generic;

[ScriptService]
public class CascadingDropdown3 : System.Web.Services.WebService
{

  [WebMethod]
  public CascadingDropDownNameValue[] GetVendors(string
   knownCategoryValues, string category)
  {
    List<CascadingDropDownNameValue> l = new
   List<CascadingDropDownNameValue>();

    l.Add(new CascadingDropDownNameValue(
      "International", "1"));
    l.Add(new CascadingDropDownNameValue(
      "Electronic Bike Repairs & Supplies", "2"));
    l.Add(new CascadingDropDownNameValue(
      "Premier Sport, Inc.", "3"));
    return l.ToArray();
  }

}
```

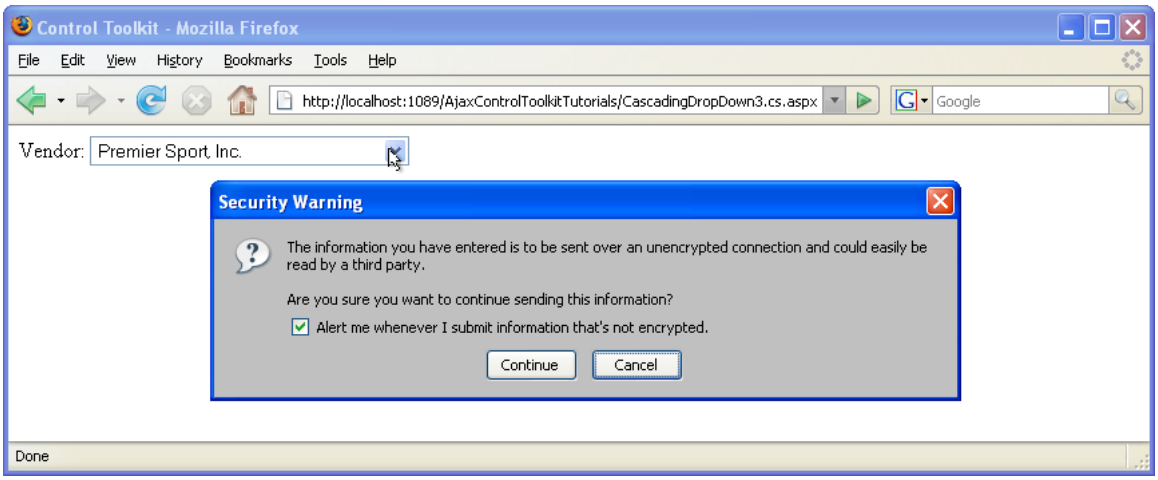
Loading the page in the browser will fill the dropdown list with three vendors, the second one being preselected. Also, ASP.NET defines the **__doPostBack()** JavaScript method. Once the page has been loaded, this JavaScript call is added to the dropdown list, but only if there are elements in it. If there are no elements in the list, the Control Toolkit is currently loading them, so the JavaScript code uses a timeout and tries again in a half second.

```
<script type="text/javascript">
  function pageLoad()
  {
    addAutoPostBack();
  }
  function addAutoPostBack()
  {
    if ($get("VendorsList").options.length > 0)
   {
      $get("VendorsList").setAttribute(
        "onchange",

   "javascript:setTimeout('__doPostBack(\\'VendorsList\\',\\'\\')'
   , 0)"
      );
    }
    else
    {
      setTimeout("addAutoPostBack()", 500);
    }
```

```
  }
</script>
```

This way, a postback is only executed when there are actually elements in the list and the user selects an entry.



Selecting a list element causes a postback