Use LINQ to XML Instead of a Database

When people think of having to store data for their applications, a database such as SQL Server immediately comes to mind. However, XML files are very handy for storing data without the overhead of a database. Using XML files to cache often-used, but seldom changed data such as US state codes, Country code, employee types and other validation tables can avoid network round-trips to a database, and potentially speed up your application. In addition, XML files are great for off-line applications where a user needs to add, edit and delete data when they can't connect to a database.

To take advantage of XML within your application, you should use LINQ to XML. I am sure you have used LINQ to iterate over other types of collections. LINQ works great to iterate over XML, too. In this blog post you will use LINQ to XML to read state codes from an XML file and display those values on a web page.

The US State Codes XML File

An XML data file called USStates.xml is in the \Xml folder of the MVC project that you can download (see the instructions at the end of this post). This XML file contains several elements of US state code data. Each row has 3 elements as shown below

```
<USStates>
  <USState>
    <StateCode>AK</StateCode>
    <StateName>Alaska</StateName>
    <Capital>Juneau</Capital>
  </USState>
</USStates>
```

The USState Class

Just as you create an Entity class to map each column in a table to a property in a class, you should do the same for an XML file too. In this case you will create a USState class with properties for each of the elements within the parent element.

```
public class USState
{
  public string StateCode { get; set; }
  public string StateName { get; set; }
  public string Capital { get; set; }
}
```

The USStateManager Class

To read the US state codes from the XML file, make sure you have a reference to the System.Xml.Linq.dll in your project. Create a class called USStateManager. This manager class is going to have a method named GetStateCodes to which you pass in the full path and file name where the USStates.xml file is located in your MVC project. The full class listing is shown below. Take a look at the code, then we will discuss each piece.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Xml.Linq;

namespace LinqToXmlSample
{
  public class USStateManager
  {
    public List<USState> GetStateCodes(string fileName) {

      // Load the XML file
      XElement xelem = XElement.Load(fileName);

      // Fill a list of USState objects
      var coll = from elem in xelem.Descendants("USState")
        select new USState
        {
          StateCode = GetValue<string>(elem.Element("StateCode"),
                                       ""),
          StateName = GetValue<string>(elem.Element("StateName"),
                                       ""),
          Capital = GetValue<string>(elem.Element("Capital"), "")
        };

      return coll.ToList();
    }

    private T GetValue<T>(XElement elem, T defaultValue) {
      if (elem == null || string.IsNullOrEmpty(elem.Value)) {
        return defaultValue;
      }
      else {
        return (T)Convert.ChangeType(elem.Value, typeof(T)); ;
      }
    }
  }
}
```

At the top of this file, you need a few using statements to support the various classes you use for reading in the XML and creating a list of USState objects. The GetStateCodes method accepts a fully qualified path and file name to the US state codes xml file. Using the XElement class, pass the file name to the Load() method. This method loads the XML file and returns an XElement object.

Once the XML file has been loaded, you write a LINQ query to iterate over the “USState” Descendants in the XML file. The “select” portion of the LINQ query creates a new USState object for each row in the XML file. You retrieve each element by passing each element name to the Element() method. This method returns an XElement object. This object has Value property from which you can retrieve the value, or a null value if the element does not exist.

Because you do not want to have a null value returned, a GetValue() method is written to accept an XElement object and a default value. This generic method checks to make sure that the XElement is not null, or that the value in the XElement object is not null or an empty string. If it is, then the default value you

pass in is returned. If there is a value in the element, then that value is converted into the specified data type.

Calling the GetStateCodes Method

Now that you have written the code to retrieve the state codes and have them returned in a list of USState objects, let's now call this method from a controller and display the state codes in a unordered list on an HTML page. Create a MVC controller named USStateController. Write the following code to create an instance of the USStateManager class, get the path and file name to the USStates.xml file, and call the GetStateCodes method.

```
public class USStateController : Controller
{
  public ActionResult Index() {
    USStateManager mgr = new USStateManager();

    string fileName = Server.MapPath("/Xml") + @"\USStates.xml";

    return View(mgr.GetStateCodes(fileName));
  }
}
```

Once you call the GetStateCodes method, the list of USState objects is passed directly to the CSHTML page via the View() method. The CSHTML page looks like the following. This page loops through each state code in the Model and builds a simple unordered list.

```
@model List<USState>

@{
  ViewBag.Title = "US State Codes";
}

<div class="row">
  <div class="col-md-12">
    <ul>
      @foreach (USState state in Model) {
        <li>
          @state.StateName (@state.StateCode)
        </li>
      }
    </ul>
  </div>
</div>
```

Summary

XML files can be a great alternative to storing data in a SQL Server table. LINQ to XML is a great way to retrieve data from those XML files, and if the files are stored on the web server, they can increase performance because you don't have to go across the network to a database server. I have used LINQ to XML to work with XML files up to 10,000 rows of data and have seen a less than a second response times. I am sure you will find a lot of uses for XML in your applications.

Sample Code

You can download the complete sample code at my website. http://www.pdsa.com/downloads. Choose "PDSA Blog", then " Use LINQ to XML Instead of a Database" from the drop-down.