## Data Access – Server Side

**Lab Time**: 45 minutes

**Lab Folder**: C:\Student\Labs\DataAccessServerSide

**Lab Overview**: LINQ to SharePoint is a technology for querying SharePoint lists that relieves the developer from having to write CAML queries. In this lab, you will be making use of the new support for LINQ in SharePoint. In the first exercise, you will be creating lists for use with LINQ, and the second exercise will create a web part for accessing the list data using LINQ.

Lab Setup Requirements

* Before you begin this lab, you must run the batch file named **SetupLab.bat**. This batch file creates a new blank site collection at the location **http://intranet.wingtip.com/sites/ServerData**. This is the site you will use to test and debug the code you are going to write with the Visual Studio 2010 SharePoint Tools.

### Exercise 1: Creating List Data

In this exercise you will create a feature to provision lists. Because LINQ code is tied to specific list schemas, your solutions will often contain a list-provisioning component.

1. Launch the Internet Explorer and navigate to the top-level site at **http://intranet.wingtip.com/sites/ServerData**. Take a moment to inspect the site and make sure it behaves as expected. Note that the setup script creates a new site collection with a Team site as its top-level site.
2. Launch **Visual Studio 2010** and create a new **Empty SharePoint Project** by selecting **File » New » Project** and give it a name of **LINQLists**.
3. Complete the SharePoint Customization wizard that appears using the following information.

**Debugging site:** http://intranet.wingtip.com/sites/ServerData

**Deploy as a farm solution**: selected

1. When the new project is created, right click the **Feature** node and choose to add a new feature.
2. Right-click the new feature and select **Add Event Receiver** to add a Feature Receiver.
3. Open the Feature1EventReceiver.cs file in Visual Studio for editing.
4. Add the following code to the feature receiver class to help with the creation of fields in the lists. Note that you can copy and paste the text from the Snip1.txt file in this lab’s **[[LAB FILES]]\StarterFiles** directory.

using System;

using System.Runtime.InteropServices;

using System.Security.Permissions;

using Microsoft.SharePoint;

using Microsoft.SharePoint.Security;

namespace LINQLists.Features.Feature1 {

[Guid("a5bcd625-e0b9-4126-b544-89b478334be0")]

public class Feature1EventReceiver : SPFeatureReceiver {

private void FixupField(SPList spList, string fieldInternalName) {

FixupField(spList.Fields.GetFieldByInternalName(fieldInternalName));

}

private void FixupField(SPField spField) {

// This method takes an InternalName of a field in a spList

// and process a few things we want all fields to have by default

// for example setting them to show into the default view

spField.ShowInDisplayForm = true;

spField.ShowInEditForm = true;

spField.ShowInListSettings = true;

spField.ShowInNewForm = true;

spField.ShowInVersionHistory = true;

spField.ShowInViewForms = true;

// Add field to default view

SPView defaultView = spField.ParentList.DefaultView;

defaultView.ViewFields.Add(spField);

defaultView.Update();

spField.Update();

}

1. Locate and uncomment the FeatureActivated() method. This is the method that will run when your feature is activated in a SharePoint site.
2. Add the following code to the FeatureActivated() method to build a number of lists. Note that you can copy and paste this code from the snip2.txt file in this lab’s **[[LAB FILES]]\StarterFiles** directory.

public override void FeatureActivated(SPFeatureReceiverProperties properties) {

SPWeb spWeb = properties.Feature.Parent as SPWeb;

//Projects List

Guid pListGuid = spWeb.Lists.Add("Projects",

"Company Projects",

SPListTemplateType.GenericList);

spWeb.Update();

//Projects List columns

SPList pList = spWeb.Lists[pListGuid];

pList.OnQuickLaunch = true;

SPField pTitleIDField = pList.Fields["Title"];

FixupField(pList, pList.Fields.Add("Description", SPFieldType.Text, false));

FixupField(pList, pList.Fields.Add("Due Date", SPFieldType.DateTime, false));

SPFieldDateTime dueDateField = (SPFieldDateTime)pList.Fields["Due Date"];

dueDateField.DisplayFormat = SPDateTimeFieldFormatType.DateOnly;

dueDateField.Update();

pList.Update();

// Employees List

Guid eListGuid = spWeb.Lists.Add("Employees",

"Employees",

SPListTemplateType.GenericList);

spWeb.Update();

// Employees List columns

SPList eList = spWeb.Lists[eListGuid];

eList.OnQuickLaunch = true;

SPField titleIDField = eList.Fields["Title"];

titleIDField.Title = "Fullname";

titleIDField.Update();

FixupField(eList, eList.Fields.Add("JobTitle", SPFieldType.Text, false));

FixupField(eList, eList.Fields.Add("Team", SPFieldType.Text, false));

FixupField(eList, eList.Fields.Add("Contribution (in Milestones)",

SPFieldType.Number, false));

string projectFieldInternalName = eList.Fields.AddLookup("Project", pListGuid, false);

SPFieldLookup projectField =

(SPFieldLookup)eList.Fields.GetFieldByInternalName(projectFieldInternalName);

projectField.LookupField = pTitleIDField.InternalName;

FixupField(projectField);

eList.Update();

// Project Manager field (Project to Employee lookup)

string employeeFieldInternalName =

pList.Fields.AddLookup("Primary Contact", eListGuid, false);

SPFieldLookup managerField =

(SPFieldLookup)pList.Fields.GetFieldByInternalName(employeeFieldInternalName);

managerField.LookupField = titleIDField.InternalName;

FixupField(managerField);

pList.Update();

}

1. Locate and uncomment the FeatureDeactivating() method. This is the method that will run when your feature is being deactivated in a SharePoint site.
2. Add the following code to the FeatureDectivating() method to tear down the lists.

public override void FeatureDeactivating(SPFeatureReceiverProperties properties)

{

SPWeb spWeb = properties.Feature.Parent as SPWeb;

SPList empList = spWeb.Lists["Employees"];

empList.Delete();

spWeb.Update();

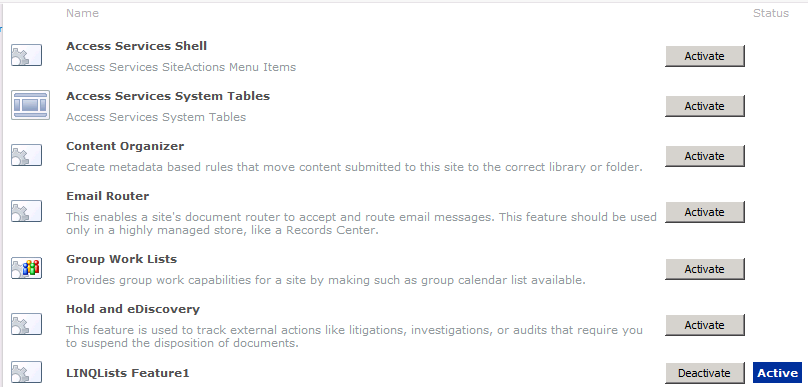
SPList projList = spWeb.Lists["Projects"];

projList.Delete();

spWeb.Update();

}

1. Build the project and verify that the code is correct; fixing any errors that may be reported by the compiler.
2. Once the project is completed, select **Debug » Start Without Debugging** from the Visual Studio main menu or use the shortcut key combination of or by pressing **[CTRL]+[F5]**. Note that when you run the project from Visual Studio, the project is built, packaged, and deployed, automatically. After the project runs, your browser will open to the test site specified at the beginning of the exercise.
3. When the test site opens, select **Site Actions » Site Settings**.
4. On the **Site Settings** page, under the **Site Actions** section select **Manage site features**.
5. On the **Site Features** page, verify that the **LINQLists Feature1** is activated.



1. After the feature is activated, you will see two new lists on the Quick Launch bar: **Projects** and **Employees**. Add some names to the **Employees** list and then add some items to the **Projects** list. The **Projects** list and **Employees** list have mutual lookups, so you’ll want to flip back and forth between the lists to add items.

In this exercise you created two lists and added some data to them for future exercises.

### Exercise 2: Creating Entities using the SPMetal Utility

In this exercise you will create entities for use with LINQ. Entity creation is done by using the command line utility SPMetal.

1. Open a command window and navigate to the following path.

c:\Program Files\Common Files\Microsoft Shared\web server extensions\14\bin

1. At the command prompt, execute the following command to build a set of entity classes.

SPMetal /web:http://intranet.wingtip.com/sites/ServerData /code:Entities.cs /language:csharp

1. Locate the file named Entities.cs that was created in the previous step. By default the file will be placed in the same location where SPMetal was executed (in this case, the SharePoint Root folder).
2. Open the file using Notepad or Visual Studio and examine the code inside that was automatically generated by the SPMetal utility. This file will be used in the next exercise to leverage the new SharePoint LINQ capabilities.

In this exercise you used the SPMetal utility to create a class that is used as the underlying source for SharePoint LINQ queries.

### Exercise 3: Creating a Web Part that uses LINQ

In this exercise you will create a Web Part that queries the lists you created earlier. The Web Part will create a view of the lists.

1. In the **Visual Studio 2010**, create a new **Visual Web Part** project named **LINQListsPart**.
2. Complete the wizard that appears using the following information.

**Debugging site:** http://intranet.wingtip.com/sites/ServerData

**Deploy as a farm solution**: selected

1. Add the Entities.cs file you created in the previous exercise to the project by right-clicking the project and selecting **Add » Existing Item**.
2. Open the Entities.cs file and scroll through the classes. You’ll notice that a DataContext has been defined along with classes for all of the list content types in your site. Ignore all the code errors in the file… these are appearing because the project is missing a needed reference.
3. To work with LINQ in SharePoint, you need to add a reference to a new assembly. Do this with the following steps:
   1. Select **Project » Add Reference** from the Visual Studio main menu.
   2. In the **Add Reference** dialog, browse to c:\Program Files\Common Files\Microsoft Shared\web server extensions\14\ISAPI and select Microsoft.SharePoint.Linq.dll. Then click the **OK** button.

Add a Literal control named display to the design surface of VisualWebPart1UserControl.ascx**.**

**<asp:Literal ID=”Display” runat=”server” />**

1. Add the following statement to the top of the VisualWebPart1UserControl.ascx.cs file to reference the necessary assemblies:

using System;

using System.Linq;

using System.Text;

using Microsoft.SharePoint;

using Microsoft.SharePoint.Linq;

using System.Web.UI;

using System.Web.UI.WebControls;

using System.Web.UI.WebControls.WebParts;

1. Add the following code to the Page\_Load() method of the Web Part that will query the SharePoint site using LINQ thanks to the Entities.cs code file you generated using SPMetal. Note that you can copy and paste this text from the snip3.txt file in this lab’s **[[LAB FILES]]\StarterFiles** directory.

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

{

StringBuilder writer = new StringBuilder();

try {

using (EntitiesDataContext dc =

new EntitiesDataContext("http://intranet.wingtip.com/sites/ServerData")) {

//Query Expressions

var q = from emp in dc.Employees

where emp.Project.DueDate < DateTime.Now.AddYears(5)

orderby emp.Project.DueDate

select new { emp.Title, Contact = emp.Project.PrimaryContact.Title };

writer.Append("<table border=\"1\" cellpadding=\"3\" cellspacing=\"3\">");

foreach (var employee in q) {

writer.Append("<tr><td>");

writer.Append(employee.Title);

writer.Append("</td><td>");

writer.Append(employee.Contact);

writer.Append("</td></tr>");

}

}

}

catch (Exception x) {

writer.Append("<tr><td>");

writer.Append(x.Message);

writer.Append("</td></tr>");

}

finally {

writer.Append("</table>");

Display.Text = writer.ToString();

}

}

1. Build the project and verify that the code compiles.
2. Set a breakpoint in the Page\_Load() method and select **Debug » Start Debugging**.
3. The Visual Studio debugger should launch a browser and navigate to the **http://intranet.wingtip.com/sites/ServerData** site.
4. Put the home page in edit mode by selecting **Site Actions » Edit Page**.
5. Click the shaded box in a Web Part Zone that says **Add Web Part**.
6. Your new Web Part will be in the **Custom** category. Locate the Web Part **VisualWebPart1**, select it and then click the **Add** button.
7. When the Web Part is added to the page, your breakpoint should be hit. Step through the code and verify that it is working correctly.

In this exercise you created a visual Web Part that used SharePoint LINQ to query a list and display the results.