Microsoft

OFFICIAL MICROSOFT LEARNING PRODUCT

10175A

Lab Instructions and Lab Answer Key: Microsoft® SharePoint® 2010, Application Development

Product Number: 10175A

Part Number: X17-41913

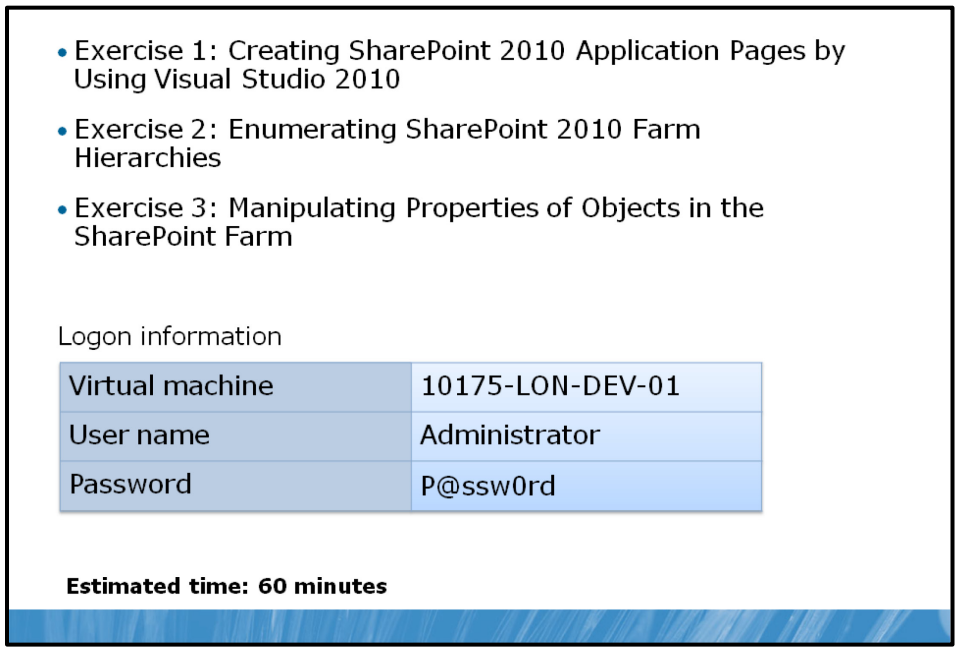
Released: 06/2010

Module 1

Lab Instructions: Introduction to the SharePoint 2010 Development Platform

Contents:

Lab: Developing with the SharePoint 2010 Object Hierarchy



Log On to the Virtual Machine for This Lab

For this lab, use the available virtual machine environment. Before you begin the lab, log on to the **10175-LON-DEV-01** virtual machine as **Administrator**, with a password of **P@ssw0rd**.

Exercise 1: Creating SharePoint 2010 Application Pages by Using Visual Studio 2010

Scenario

In this exercise, you will create and design the user interface for two SharePoint 2010 application pages that work together. The first application page will contain a tree view for displaying the entire farm hierarchy, including services, Web applications, site collections, Webs, subwebs, and lists. The second page will enable the user to set specific properties for Webs or lists.

You implement the user interfaces in this exercise, and you then implement the code-based logic in subsequent exercises.

The main tasks for this exercise are as follows:

1. Create a Visual Studio 2010 project for SharePoint.
2. Add two application pages to the SharePoint 2010 project.
3. Design the user interfaces for the SharePoint application pages.

▶ Task 1: Create a Visual Studio 2010 project for SharePoint

1. On the **Start** menu, click **All Programs**, click **Microsoft Visual Studio 2010** and then click **Microsoft Visual Studio 2010**.
2. On the **File** menu, point to **New**, and then click **Project**.
3. In the **Installed Templates** section, expand the **Visual C#** node, expand the **SharePoint** node and then click **2010**.
4. Click **Empty SharePoint Project**.
5. In the **Name** text box, type **Lab01**.
6. In the **Location** text box, type **E:\Student\Lab01\Starter**.
7. Click **OK**.

   The SharePoint Customization Wizard appears.
8. In the **What local site do you want to use for debugging?** text box, type **http://sharepoint**.
9. In the **What is the trust level for this SharePoint** solution section, click the **Deploy as a farm solution** option.
10. Click **Finish**.

    The project is created.

▶ Task 2: Add Application pages to the SharePoint 2010 project

1. In the Solution Explorer window, right-click **Lab01**, point to **Add** and then click **New Item**.
2. Click **Application Page**.
3. In the **Name** text box, type **FarmHierarchy.aspx** and then click **Add**.
4. In the Solution Explorer window, right-click **Lab01**, point to **Add** and then click **New Item**.
5. Click **Application Page**.
6. In the **Name** text box, type **PropertyChanger.aspx** and then click **Add**.

Task 3: Add user interface components to SharePoint Application pages

1. Click the **FarmHierarchy.aspx** tab.
2. Add the following markup between the opening and closing tags of the **<asp:Content>** element that has an **ID** of **Main**:

```
<h2>My Farm</h2>
<asp:TreeView ID="farmHierarchyViewer" runat="server"
  ShowLines="true" EnableViewState="true">
</asp:TreeView>
```

3. Change the markup between the opening and closing tags of the **<asp:Content>** element that has an **ID** of **PageTitle** to the following:

```
Farm Hierarchy and Properties
```

4. Change the markup between the opening and closing tags of the **<asp:Content>** element that has an **ID** of **PageTitleInTitleArea** to:

```
My Farm Hierarchy
```

5. Click the **PropertyChanger.aspx** tab.
6. Add the following markup between the opening and closing tags of the **<asp:Content>** element that has an **ID** of **Main**:

```
<h2>Properties:</h2>
<asp:Label ID="objectName" runat="server" Text="">
</asp:Label><br/><br/>
<asp:Panel ID="webProperties" runat="server" Visible="false"
  BorderColor="Orange" BorderStyle="Dashed" BorderWidth="1">
  <asp:Label ID="WebLabel" runat="server" Text="Web Title">
  </asp:Label><br/>
  <asp:TextBox ID="webTitle" runat="server" EnableViewState="true">
  </asp:TextBox> 
  <asp:Button ID="webTitleUpdate" runat="server" Text="Update"/>
   
  <asp:Button ID="webCancel" runat="server" Text="Cancel" />
</asp:Panel>
<asp:Panel ID="listProperties" runat="server" Visible="false"
  BorderColor="Orange" BorderStyle="Dashed" BorderWidth="1">
  <asp:Label ID="ListLabel" runat="server" Text="List Properties">
  </asp:Label><br/>
  <asp:CheckBox ID="listVersioning" runat="server"
    EnableViewState="true" Text="Enable Versioning" />
  <br/>
  <asp:CheckBox ID="listContentTypes" runat="server"
    EnableViewState="true" Text="Enable Content Types" />
   
  <asp:Button ID="listPropertiesUpdate" runat="server"
    Text="Update" />
   
  <asp:Button ID="listCancel" runat="server" Text="Cancel"/>
</asp:Panel>
```

7. Change the markup between the opening and closing tags of the **<asp:Content>** element that has an **ID** of **PageTitle** so that it reads:

```
Property Changer
```

8. Change the markup between the opening and closing tags of the **<asp:Content>** element that has an **ID** of **PageTitleInTitleArea** so that it reads:

```
My Property Editor
```

9. On the **File** menu, click **Save All**.

Results: After this exercise, you should have created a Visual Studio project that contains two application pages. You should also have added user interface components to those pages.

Exercise 2: Enumerating SharePoint 2010 Farm Hierarchies

Scenario

In this exercise, you iterate through the services, Web applications, site collections, Webs, and lists in the SharePoint Farm, and you add details for each of these objects to the tree-view control. Furthermore, you implement the tree-nodes for Webs and for Lists so that when a user clicks one of those nodes, they navigate to the PropertyChanger.aspx file (which you created previously) so that they can manage specific properties.

The main task for this exercise is as follows:

- Populating a tree-view control from object properties in a SharePoint Farm, and ensuring that the hierarchical nature of those objects is reflected in the tree-view structure.

▶ Task 1: Retrieve details of services and Web applications from the SharePoint farm

1. In the Solution Explorer window, right-click **FarmHierarchy.aspx**, and click **View Code**.
2. Near the top of the file, directly beneath the existing **using** statements, add the following code:

```
using System.Web.UI.WebControls;
using Microsoft.SharePoint.Administration;
```

3. Between the opening and closing braces of the **Page_Load** function, add the following code:

```
SPFarm thisFarm = SPFarm.Local;
TreeNode node;
farmHierarchyViewer.Nodes.Clear();
foreach (SPService svc in thisFarm.Services)
{
  node = new TreeNode();
  node.Text = "Farm Service (Type=" + svc.TypeName + "; Status="
    + svc.Status + ")";
  farmHierarchyViewer.Nodes.Add(node);
  TreeNode svcNode = node;
  if (svc is SPWebService)
  {
    SPWebService webSvc = (SPWebService)svc;
    foreach (SPWebApplication webApp in webSvc.WebApplications)
    {
      node = new TreeNode();
      node.Text = webApp.DisplayName;
      svcNode.ChildNodes.Add(node);
      TreeNode webAppNode = node;
      if (!webApp.IsAdministrationWebApplication)
      {
        foreach (SPSite site in webApp.Sites)
        {
          site.CatchAccessDeniedException = false;
          try
          {
            node = new TreeNode();
            node.Text = site.Url;
            webAppNode.ChildNodes.Add(node);
            TreeNode siteNode = node;
            node = new TreeNode(site.RootWeb.Title, null, null,
              site.RootWeb.Url +
"/_layouts/lab01/PropertyChanger.aspx?type=web&objectID="
              + site.RootWeb.ID, "_self");
            siteNode.ChildNodes.Add(node);
            TreeNode parentNode = node;
            foreach (SPList list in site.RootWeb.Lists)
            {
```

```
              node = new TreeNode(list.Title, null, null,
                site.RootWeb.Url +
"/_layouts/lab01/PropertyChanger.aspx?type=list&objectID="
                + list.ID, "_self");
              parentNode.ChildNodes.Add(node);
            }
            foreach (SPWeb childWeb in site.RootWeb.Webs)
            {
              try
              {
                addWebs(childWeb, parentNode);
              }
              finally
              {
                childWeb.Dispose();
              }
            }
            site.CatchAccessDeniedException = false;
          }
          finally
          {
            site.Dispose();
          }
        }
      }
    }
  }
}
farmHierarchyViewer.ExpandAll();
```

4. Below the closing brace for the **Page_Load** function, add the following code:

```
void addWebs(SPWeb web, TreeNode parentNode)
{
  TreeNode node;
  node = new TreeNode(web.Title, null, null, web.Url
    + "/_layouts/lab01/PropertyChanger.aspx?type=web&objectID="
    + web.ID, "_self");
  parentNode.ChildNodes.Add(node);
  parentNode = node;
  foreach (SPList list in web.Lists)
  {
    node = new TreeNode(list.Title, null, null, web.Url
      + "/_layouts/lab01/PropertyChanger.aspx?type=list&objectID="
      + list.ID, "_self");
    parentNode.ChildNodes.Add(node);
  }
  foreach (SPWeb childWeb in web.Webs)
  {
    try
    {
      addWebs(childWeb, parentNode);
    }
    finally
    {
      childWeb.Dispose();
    }
  }
}
```

5. On the **File** menu, click **Save All**.

Results: After this exercise, you should have written code that loads the entire farm hierarchy into the tree-view control, and you should have provided Urls for Web and list objects that can be navigated to when the user clicks a Web or list in the tree-view control.

Exercise 3: Manipulating Properties of Objects in the SharePoint Farm

Scenario

In this exercise, you determine whether the PropertyChanger.aspx page was loaded by a user clicking a Web or list object in the tree-view you created in the previous exercise. Depending on which type of object was clicked, you manipulate the user interface so that the user can review and set properties for a specific object, as appropriate. Your code makes changes to those objects in response to the user's actions.

The main tasks for this exercise are as follows:

1. Showing either the webProperties panel or the listProperties panel in the PropertyChanger.aspx page, depending on query string parameters, and populating user interface controls with values extracted from the properties of SharePoint objects.
2. Updating properties of SharePoint objects based on values supplied by the user.
3. Testing the entire solution.

▶ Task 1: Retrieve properties of SharePoint objects

1. In the Solution Explorer window, right-click **PropertyChanger.aspx**, and click **View Code**.
2. Directly above the **Page_Load** function, add the following code:

```
SPWeb thisWeb = null;
SPList thisList = null;
```

3. Between the opening and closing braces of the **Page_Load** function, add the following code:

```
webTitleUpdate.Click += new EventHandler(webTitleUpdate_Click);
listPropertiesUpdate.Click +=
    new EventHandler(listPropertiesUpdate_Click);
webCancel.Click += new EventHandler(allCancel_Click);
listCancel.Click += new EventHandler(allCancel_Click);
try
{
  string objectType = string.Empty;
  string objectID = string.Empty;
  string objectUrl = string.Empty;
  if (this.Page.Request["type"] != null)
  {
    objectType = this.Page.Request["type"].ToString();
  }
  else
  {
    objectName.Text = "Malformed URL";
    listProperties.Visible = false;
    webProperties.Visible = false;
    return;
  }
  if (this.Page.Request["objectID"] != null)
  {
    objectID = this.Page.Request["objectID"].ToString();
  }
  else
  {
    objectName.Text = "Malformed URL";
    listProperties.Visible = false;
    webProperties.Visible = false;
    return;
  }
  if (objectType == "web")
```

```
  {
    listProperties.Visible = false;
    webProperties.Visible = true;
    SPSite thisSite = SPContext.Current.Site;
    thisWeb = thisSite.OpenWeb(new Guid(objectID));
    objectName.Text = "Web: " + thisWeb.Title;
    if (!Page.IsPostBack)
    {
      webTitle.Text = thisWeb.Title;
      thisWeb.Dispose();
      //Do NOT dispose of thisSite!
    }
  }
  if (objectType == "list")
  {
    webProperties.Visible = false;
    listProperties.Visible = true;
    thisWeb = SPContext.Current.Web;
    thisList = thisWeb.Lists[new Guid(objectID)];
    objectName.Text = "List: " + thisList.Title;
    if (!Page.IsPostBack)
    {
      listVersioning.Checked = thisList.EnableVersioning;
      listContentTypes.Checked = thisList.ContentTypesEnabled;
      //Do NOT dispose of thisWeb!
    }
  }
}
catch (Exception ex)
{
  objectName.Text = ex.Message;
}
```

4. On the **File** menu, click **Save All**.

▶ Task 2: Set properties of SharePoint objects

1. Below the closing brace for the **Page_Load** function, add the following code:

```
void webTitleUpdate_Click(object sender, EventArgs e)
{
  try
  {
    thisWeb.AllowUnsafeUpdates = true;
    thisWeb.Title = webTitle.Text;
    thisWeb.Update();
    thisWeb.AllowUnsafeUpdates = false;
    this.Page.Response.Redirect(thisWeb.Url
      + "/_layouts/lab01/FarmHierarchy.aspx");
  }
  catch(Exception ex)
  {
    objectName.Text = ex.Message;
    listProperties.Visible = false;
    webProperties.Visible = false;
  }
}
```

2. Below the closing brace for the **webTitleUpdate_Click** function, add the following code:

```
void listPropertiesUpdate_Click(object sender, EventArgs e)
{
  try
  {
    thisWeb.AllowUnsafeUpdates = true;
```

```
    thisList.EnableVersioning = listVersioning.Checked;
    thisList.ContentTypesEnabled = listContentTypes.Checked;
    thisList.Update();
    thisWeb.AllowUnsafeUpdates = false;
    this.Page.Response.Redirect(thisWeb.Url
      + "/_layouts/lab01/FarmHierarchy.aspx");
  }
  catch (Exception ex)
  {
     objectName.Text = ex.Message;
     listProperties.Visible = false;
     webProperties.Visible = false;
  }
}
```

3. Below the closing brace for the **listPropertiesUpdate_Click** function, add the following code:

```
void allCancel_Click(object sender, EventArgs e)
{
  this.Page.Response.Redirect(thisWeb.Url
    + "/_layouts/lab01/FarmHierarchy.aspx");
}
```

4. On the **File** menu, click **Save All**.

Task 3: Test the solution

1. In the Solution Explorer window, right-click **Lab01** and then click **Deploy**.
2. On the **Start** menu, click **Internet Explorer**.
3. In the **Address Bar**, type **http://sharepoint/_layouts/lab01/farmhierarchy.aspx** and press ENTER.
4. When the page has loaded, review the details shown in the tree-view control.
5. In the tree-view control, click the **Home** node.

   The PropertyChanger.aspx page appears.
6. In the **Web Title** text box, type **HR Intranet** and then click **Update**.

   The title of the Web in the breadcrumb control and in the tree-view control is updated to reflect your changes.
7. In the tree-view control, click the **Shared Documents** node.
8. Check the **Enable Versioning** check box.
9. Check the **Enable Content Types** check box.
10. Click **Update**.
11. In the tree-view control, click the **Tasks** node.
12. Clear the **Enable Content Types** check box.
13. Click **Update**.
14. In the tree-view control, click the **Tasks** node once more.

    The Tasks list currently does not support content types, because your code set this property to false when you clicked Update in step 13.
15. Check the **Enable Content Types** check box.

16. Click **Update**.
17. Close all applications. You have now completed this lab.

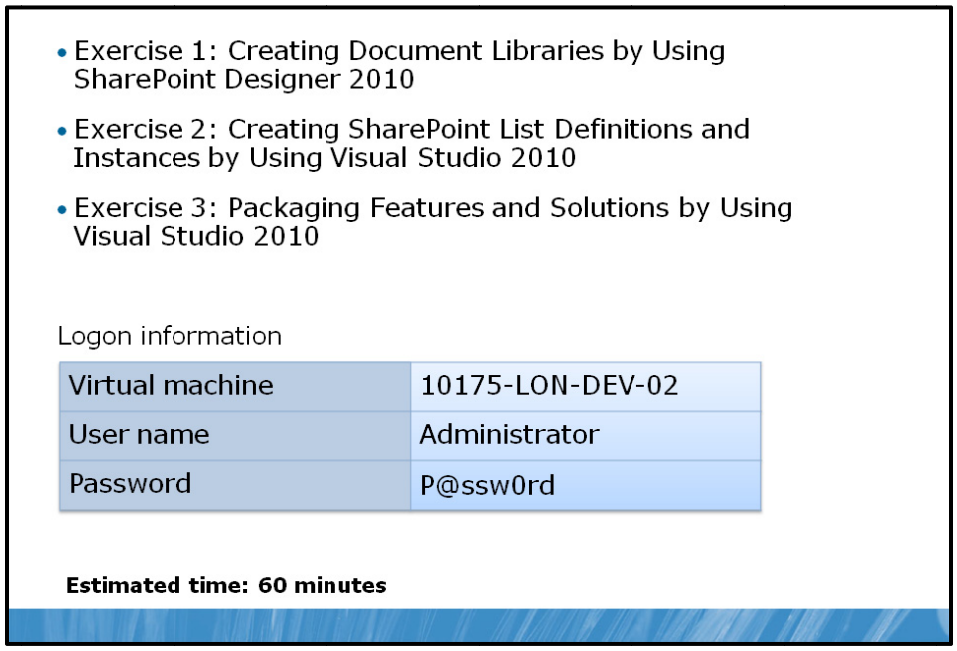
Results: After this exercise, you should have added code to show either the webProperties panel or the listProperties panel in the PropertyChanger.aspx page, depending on query string parameters, and you should have populated user interface controls with values extracted from the properties of SharePoint objects. You should also have written code that updates properties of SharePoint objects based on values supplied by the user, and finally you should have tested the entire solution.

Module 2

Lab Instructions: Using SharePoint 2010 Developer Tools

Contents:

Lab: Using SharePoint 2010 Developer Tools



Log On to the Virtual Machine for This Lab

For this lab, you use the available virtual machine environment. Before you begin the lab, you must log on to the **10175-LON-DEV-02** virtual machine as **Administrator**, with a password of **P@ssw0rd**.

Exercise 1: Creating Document Libraries by Using SharePoint Designer 2010

Scenario

In this exercise, you create and design a Resumes document library where resumes from job applicants can be stored. You add a column to indicate whether the applicant is currently active, and you define a view to show only resumes for active applicants.

The main tasks for this exercise are as follows:

1. Editing the HR Intranet Site in SharePoint Designer 2010 and adding a new library.
2. Adding a column to the library.
3. Creating a view for the library.
4. Testing the new library.

▶ Task 1: Edit the HR Intranet site in SharePoint Designer 2010

1. On the Start menu, click Internet Explorer.
   The Home page for the HR Intranet site appears in Windows Internet Explorer®.
2. On the **Site Actions** menu, click **Edit in SharePoint Designer**.
   Microsoft SharePoint Designer appears, and the site is opened.
3. On the ribbon, click Document Library, and then click Document Library.
   The Create list or document library dialog box appears.
4. In the Name text box, type Resumes.
5. In the **Description** text box, type **Library for storing resumes from job applicants**, and then click **OK**.

▶ Task 2: Add a column to the Resumes library

1. In the Site Objects pane, click **Lists and Libraries**.
2. In the **Document Libraries** section, click **Resumes**.
3. In the **Customization** section, click **Edit list columns**.
4. On the ribbon, click **Add New Column**, and then click **Yes/No (checkbox)**.
5. Type **Active** and press ENTER.
6. On the **Quick Access** toolbar, click **Save**.

▶ Task 3: Create a view for the Resumes library

1. In the breadcrumb control, click **Resumes**.
2. In the **Views** section, click **New**.
   The Create New List View dialog box appears.
3. In the **Name** text box, type **Active**, and then click **OK**.
4. In the **Views** section, click **Active** and wait for the page to load.
5. Click the **Design** tab and then click **Type**.
   The PlaceholderMain (Custom) control is highlighted.
6. On the ribbon, click **Filter**.
   The Filter Criteria dialog box appears.
7. In the **Field Name** drop-down list, click **Active**.
8. In the **Comparison** drop-down list, click **Equals**.
9. In the **Value** drop-down list, click **Yes**.
10. Click **OK**.
11. On the **Quick Access** toolbar, click **Save**.
12. Close SharePoint Designer.

Task 4: Test the Resumes library

1. Switch to Internet Explorer, and in the breadcrumb control click **HR Intranet**.
2. In the **Quick Launch** bar, click **Resumes**.
   Note that the Resumes library has a column named Active.
3. In the breadcrumb control, click **All Documents** and note that the drop-down list includes a view called **Active**.
4. Close Internet Explorer.

Results: After this exercise, you should have created a new document library with a custom column and a new view.

Exercise 2: Creating SharePoint List Definitions and Instances by Using Visual Studio 2010

Scenario

In this exercise, you create a project by using Visual Studio 2010, and you add multiple list definitions and instances to the project. One list definition is based on a built-in base type of Tasks. The second list definition is based on a simple custom list, and you add data to the list instance associated with this definition.

You deploy these definitions by using Visual Studio.

Then, you add a third list definition based on a custom list, and you edit the schema and views of this definition in Visual Studio.

The main tasks for this exercise are as follows:

1. Creating a Visual Studio Project.
2. Adding two list definitions to the project.
3. Deploying the project.
4. Adding a third list definition to the project and modifying its schema.

▶ Task 1: Create a Visual Studio 2010 project for SharePoint

1. On the **Start** menu, click **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.
2. On the File menu, point to New, and then click Project.
3. In the Installed Templates section, expand the Visual C# node, expand the SharePoint node, and then click 2010.
4. Click **Empty SharePoint Project**.
5. In the Name text box, type Lab02.
6. In the **Location** text box, type **E:\Student\Lab02\Starter**.
7. Click OK.
8. The SharePoint Customization Wizard appears.
9. In the **What local site do you want to use for debugging?** text box, type **http://sharepoint**.
10. In the **What is the trust level for this SharePoint solution** section, click the **Deploy as a farm solution** option.
11. Click Finish.

    The project is created.

▶ Task 2: Add a list definition and instance based on the tasks template to the SharePoint 2010 project

1. In the Solution Explorer window, right-click **Lab02**, point to **Add**, and then click **New Item**.
2. Click **List Definition**.
3. In the **Name** text box, type **RecruitmentTasks**, and then click **Add**.
   The SharePoint Customization Wizard appears.
4. In the **What is the display name of the list definition?** text box, type **RecruitmentTasks**.
5. In the **What is the type of the list definition?** drop-down list, click **Tasks**.
6. Ensure that the **Add a list instance for this list definition** check box is selected, and then click **Finish**.

   The Elements.xml file for the RecruitmentTasks list definition is opened.

7. Replace the entire **ListTemplate** element with the following markup:

```
<ListTemplate
  Name="RecruitmentTasks"
  Type="10000"
  BaseType="0"
  OnQuickLaunch="FALSE"
  SecurityBits="11"
  Sequence="360"
  DisplayName="RecruitmentTasks"
  Description="Recruitment Tasks"
  Image="/_layouts/images/ittask.gif"
  AllowDeletion="TRUE"
  AllowEveryoneViewItems="TRUE"
  DisableAttachments="TRUE"
  NoCrawl="TRUE"
  />
```

8. In the Solution Explorer window, right-click **ListInstance1**, and then click **Rename**.
9. Type **Interviews** and press ENTER.
10. In the Solution Explorer window, expand **Interviews**.
11. Double-click the **Elements.xml** file that is a subnode of **Interviews**.
12. Replace the entire **ListInstance** element with the following markup:

```
<ListInstance Title="Interviews"
  OnQuickLaunch="TRUE"
  TemplateType="10000"
  Url="Lists/Interviews"
  Description="Interviews for External Candidates">
</ListInstance>
```

13. On the **File** menu, click **Save All**.
14. Close all open documents in Microsoft Visual Studio®, but leave the project open and leave Visual Studio running.

▶ Task 3: Add a list definition and instance based on the custom list template to the SharePoint 2010 project

1. In the Solution Explorer window, right-click **Lab02**, point to **Add**, and then click **New Item**.
2. Click **List Definition**.
3. In the **Name** text box, type **CandidateList**, and then click **Add**.
   The SharePoint Customization Wizard appears.
4. In the **What is the display name of the list definition?** text box, type **CandidateList**.
5. In the **What is the type of the list definition?** drop-down list, click **Custom List**.
6. Ensure that the **Add a list instance for this list definition** check box is selected, and then click **Finish**.

   The Elements.xml file for the CandidateList list definition is opened.

7. Replace the entire **ListTemplate** element with the following markup:

```
<ListTemplate
  Name="CandidateList"
  Type="10001"
  BaseType="0"
  OnQuickLaunch="FALSE"
  SecurityBits="11"
  Sequence="410"
  DisplayName="CandidateList"
  Description="External Candidates"
  Image="/_layouts/images/itgen.gif"
  AllowDeletion="TRUE"/>
```

8. In the Solution Explorer window, right-click **ListInstance1**, and then click **Rename**.
9. Type **Candidates** and press ENTER.
10. In the Solution Explorer window, expand **Candidates**.
11. Double-click the **Elements.xml** file that is a subnode of **Candidates**.
12. Replace the entire **ListInstance** element with the following markup:

```
<ListInstance Title="Candidates"
  OnQuickLaunch="TRUE"
  TemplateType="10001"
  Url="Lists/Candidates"
  Description="External Candidates">

</ListInstance>
```

13. On the **File** menu, click **Save All**.
14. Close all open documents in Visual Studio, but leave the project open and leave Visual Studio running.

▶ Task 4: Deploy and manage the solution from Visual Studio 2010

1. In the Solution Explorer window, right-click **Lab02**, and then click **Deploy**.
2. On the **Start** menu, click **Internet Explorer**. Note that the Quick Launch bar shows links to **Interviews** and **Candidates**; these are the list instances you have just deployed.
3. In the **Quick Launch** bar, click **Candidates**, and note that the list does not contain any data.
4. On the ribbon, click the **List** tab.

   If the Internet Explorer dialog box appears and prompts you that content from the website is being blocked, clear the **Continue to prompt when website content is blocked** checkbox, and then click **Close**.

5. On the ribbon, click **List Settings**. In the **Permissions and Management** section, note that there is a link named **Delete this list**.
6. In the breadcrumb control, click **HR Intranet**.
7. Leave Internet Explorer running and switch to Visual Studio.
8. In the Solution Explorer window, double-click **CandidateList** to open its **Elements.xml** file.
9. Edit the **AllowDeletion** attribute so that it reads:

```
AllowDeletion="FALSE"
```

10. In the Solution Explorer window, expand **CandidateList**, and then double-click **Candidates** to open its **Elements.xml** file.
11. Add the following markup *before* the closing tag of the **ListInstance** element:

```
<Data>
  <Rows>
    <Row>
      <Field Name="Title">Geert Camelbeke</Field>
    </Row>
    <Row>
      <Field Name="Title">Ivo Hamels</Field>
    </Row>
    <Row>
      <Field Name="Title">Eduardo Melo</Field>
    </Row>
    <Row>
      <Field Name="Title">Svetlana Omelchenko</Field>
    </Row>
    <Row>
      <Field Name="Title">Melanie Speckman</Field>
    </Row>
```

```
    <Row>
      <Field Name="Title">Steffen Tschimmel</Field>
    </Row>
  </Rows>
</Data>
```

12. On the **File** menu, click **Save All**.
13. In the Solution Explorer window, right-click **Lab02**, and then click **Deploy**.
14. In the **Deployment Conflicts** dialog box, check the **Do not prompt me again for these items** check box, and then click **Resolve Automatically**.
15. Switch to Internet Explorer. Note that the **Quick Launch** bar shows links to **Interviews** and **Candidates**; these are the list instances you have just redeployed.
16. In the **Quick Launch** bar, click **Candidates**, and note that the list now contains data.
17. On the ribbon, click the **List** tab.
18. On the ribbon, click **List Settings**. In the **Permissions and Management** section, note that there is no longer a link named **Delete this list**.
19. In the breadcrumb control, click **HR Intranet**.
20. Leave Internet Explorer running and switch to Visual Studio.

▶ Task 5: Define custom list schema

1. In the Solution Explorer window, right-click **Lab02**, point to **Add**, and then click **New Item**.
2. Click **List Definition**.
3. In the **Name** text box, type **InterviewOutcomes**, and then click **Add**.

   The SharePoint Customization Wizard appears.

4. In the **What is the display name of the list definition?** text box, type **InterviewOutcomes**.
5. In the **What is the type of the list definition?** drop-down list, click **Custom List**.
6. Ensure that the Add a list instance for this list definition check box is selected, and then click Finish.

   The Elements.xml file for the InterviewOutcomes list definition is opened.

7. Replace the entire **ListTemplate** element with the following markup:

```
<ListTemplate
  Name="InterviewOutcomes"
  Type="10002"
  BaseType="0"
  OnQuickLaunch="FALSE"
  SecurityBits="11"
  Sequence="410"
  DisplayName="InterviewOutcomes"
  Description="Outcomes of Interviews for External Candidates"
  Image="/_layouts/images/itgen.gif"/>
```

8. In the Solution Explorer window, right-click **ListInstance1**, and then click **Rename**.
9. Type **Outcomes** and press ENTER.
10. In the Solution Explorer window, expand **Outcomes**.
11. Double-click the **Elements.xml** file that is a subnode of **Outcomes**.
12. Replace the entire **ListInstance** element with the following markup:

```
<ListInstance Title="Outcomes"
  OnQuickLaunch="TRUE"
  TemplateType="10002"
  Url="Lists/Outcomes"
  Description="Interview Outcomes">
</ListInstance>
```

13. In the Solution Explorer window, double-click **Schema.xml** that is a subnode of **InterviewOutcomes**.

14. Place the cursor between the opening and closing tags of the **<Fields> </Fields>** element on line 10 of the file. Then, press ENTER.
15. In the space you have just created, add the following markup:

```
<Field ID="{c3a92d97-2b77-4a25-9698-3ab54874bc06}"
  Name="Candidate" Type="Lookup" DisplayName="Candidate"
  List="Lists/Candidates" ShowField="Title" Required="TRUE">
</Field>
<Field ID="{c3a92d97-2b77-4a25-9698-3ab54874bc19}"
  Name="Interviewer" Type="User" DisplayName="Interviewer"
  List="UserInfo" Required="TRUE" ShowField="ImnName"
  UserSelectionMode="PeopleAndGroups">
</Field>
<Field ID="{c3a92d97-2b77-4a25-9698-3ab54874bc81}"
  Name="Offer" Type="Boolean" DisplayName="Job Offered?"
  Required="TRUE">
</Field>
```

16. Press CTRL+F.
17. In the **Find what** text box, type **LinkTitleNoMenu**, and then click **Find Next**.

    Visual Studio finds markup that reads: <FieldRef Name=" LinkTitleNoMenu "></FieldRef>

18. Close the **Find and Replace** dialog box.
19. Place the cursor at the end of the line that Visual Studio has found, and then press ENTER.
20. Add the following markup to the space you have just created:

```
<FieldRef Name="Candidate"></FieldRef>
<FieldRef Name="Interviewer"></FieldRef>
<FieldRef Name="Offer"></FieldRef>
```

21. Press CTRL+F.
22. In the **Find what** text box, type **LinkTitle**, and then click **Find Next**.

    Visual Studio finds markup that reads: <FieldRef Name=" LinkTitle "></FieldRef>

23. Close the **Find and Replace** dialog box.
24. Place the cursor at the end of the line that Visual Studio has found, and then press ENTER.
25. Add the following markup to the space you have just created:

```
<FieldRef Name="Candidate"></FieldRef>
<FieldRef Name="Interviewer"></FieldRef>
<FieldRef Name="Offer"></FieldRef>
```

26. On the **File** menu, click **Save All**.

Results: After this exercise, you should have created and deployed list definitions and instances by using Visual Studio 2010.

Exercise 3: Packaging Features and Solutions by Using Visual Studio 2010

Scenario

In this exercise, you explore the Feature Designer and Package Designer in Visual Studio 2010. You also package a solution by using Visual Studio 2010, which you then deploy by using Windows Power shell.

The main tasks for this exercise are as follows:

1. Removing an item from a feature.
2. Creating a new feature and adding an item to it.
3. Removing features from a package, and then creating a WSP solution package.
4. Deploying and managing the WSP solution package by using Windows Power shell.
5. Finalizing the solution.
6. Testing the solution.

▶ Task 1: Remove items from a feature

1. In the Solution Explorer window, expand **Features**.
2. Double-click **Feature1**.

   The Feature Designer for Feature1 appears.

3. In the Items in the Feature pane, double-click **InterviewOutcomes**.

   InterviewOutcomes is removed from the Items in the Feature pane.

4. In the Items in the Feature pane, double-click **Outcomes**.

   Outcomes is removed from the Items in the Feature pane.

5. On the **File** menu, click **Save All**.
6. Close **Feature1**.

▶ Task 2: Create a new feature and add items to it

1. In the Solution Explorer window, right-click **Features**, and click **Add Feature**.

   The Feature Designer for Feature2 appears.

2. In the Solution Explorer window, right-click **Feature2**, and click **Rename**.
3. Type **Interviews** and press ENTER.
4. In the Items in the Solution pane, double-click **InterviewOutcomes**.

   InterviewOutcomes is added to the Items in the Feature pane.

5. In the Items in the Solution pane, double-click **Outcomes**.

   Outcomes is added to the Items in the Feature pane.

6. In the Title text box, type Recruitment Interviews.
7. In the **Description** text box, type **List definition and instance for the interview process**.
8. On the File menu, click Save All.
9. Close the Interviews feature.

▶ Task 3: Package Features

1. In the Solution Explorer window, double-click **Package**.
2. In the Items in the Package pane, double-click **Lab02 Feature1**.

   The feature is removed from the Items in the Package pane.

3. On the **File** menu, click **Save All**.
4. Close the Package Designer.
5. In the Solution Explorer window, right-click **Lab02**, and then click **Package**.

   Visual Studio builds and packages the solution, but does not deploy it.

6. Leave Visual Studio running.
7. Using Windows Explorer, navigate to the following folder:
   E:\Student\Lab02\Starter\Lab02\Lab02\bin\debug

   Visual Studio has created a solution file called Lab02.wsp.

8. Drag **Lab02.wsp** to the **Local Disk (C:)** folder.

▶ Task 4: Deploy solutions by using Windows Power shell

1. On the **Start** menu, click **All Programs**, click **Microsoft SharePoint 2010 Products**, and then click **SharePoint 2010 Management Shell**.

   The Windows Power Shell command prompt appears.

2. At the command prompt, type **Add-SP Solution C:\lab02.wsp**, and press ENTER.

   Windows Power shell displays a message to inform you that a solution named Lab02.wsp already exists. This is the package that you previously deployed from Visual Studio.

3. At the command prompt, type **Remove-SP Solution -Identity Lab02.wsp**, and press ENTER.

   Windows Power shell displays a message to inform you that the solution has already been deployed and must be retracted before it can be removed.

4. At the command prompt, type **Uninstall-SP Solution -Identity Lab02.wsp**, and press ENTER.
5. Press Y and then press ENTER.
6. At the command prompt, type **Remove-SP Solution -Identity Lab02.wsp**, and press ENTER. Press Y and then press ENTER.
7. At the command prompt, type **Add-SP Solution C:\Lab02.wsp**, and press ENTER.
8. At the command prompt, type **Install-SP Solution -Identity Lab02.wsp –GAC Deployment**, and press ENTER.
9. At the command prompt, type **Install-SP Solution -Identity Lab02.wsp –GAC Deployment –Force**, and press ENTER.
10. Close Windows Power Shell.

▶ Task 5: Finalize the solution

1. Switch to Visual Studio.
2. In the Solution Explorer window, double-click **Package**.
3. In the Items in the Solution pane, double-click **Lab02 Feature1**.

   The feature is added to the Items in the Package pane.

4. On the **File** menu, click **Save All**.
5. Close the Package Designer.
6. In the Solution Explorer window, right-click **Lab02**, and then click **Deploy**.

   A deployment error occurs because of the deployment you have just performed by using Power shell, but you simply need to redeploy one more time.

7. In the Solution Explorer window, right-click **Lab02**, and then click **Deploy**.

8. In the Deployment Conflicts dialog box, check the **Do not prompt me again for these items** check box, and then click **Resolve Automatically**.

   Visual Studio builds, packages, and deploys the solution.

9. Switch to Internet Explorer.
10. In the breadcrumb control, click **HR Intranet**.
11. On the **Site Actions** menu, click **Edit in SharePoint Designer**.

    SharePoint Designer appears, and the site is opened.

12. In the Site Objects pane, click **Lists and Libraries**.
13. In the **Document Libraries** section, click **Resumes**.
14. In the **Customization** section, click **Edit list columns**.
15. On the ribbon, click **Add New Column**, and then click **Lookup (Information Already on This Site)**.
16. In the **List or document library** drop-down list, click **Candidates**.
17. In the **Field** drop-down list, click **ID**.
18. In the **Add a column to show each of these additional fields** list, select the **Title** check box, and then click **OK**.
19. Right-click **NewColumn1**, and click **Rename**.
20. Type **Candidate ID**, and press ENTER.
21. On the **Quick Access** toolbar, click **Save**.
22. Close SharePoint Designer.

▶ Task 6: Test the solution

1. Switch to Internet Explorer.
2. In the **Quick Launch** bar, click **Resumes** and note the lookup columns that have been added to the library.
3. In the **Quick Launch** bar, click **Candidates** and note the list of candidates.
4. In the **Quick Launch** bar, click **Outcomes**.
5. On the ribbon, click the **List** tab.
6. On the ribbon, click **List Settings**.
7. In the **Views** section, click **All Items**.
8. Expand the **Inline Editing** section, and select the **Allow inline editing** check box.
9. Click **OK**.
10. Below the list columns, click the plus sign (**+**).
11. In the **Title** text box, type **Senior Developer**.
12. In the **Candidate** drop-down list, click **Melanie Speckman**.

    The list of names is retrieved by the lookup field you defined in Visual Studio.

13. In the **Interviewer** text box, type **KrishnaS**, and then click **Check Names**.

    The people-picker functionality is provided by the User field you defined in Visual Studio.

14. Select the **Job Offered?** check box.
15. Click **Save** for the list item.
16. Close all applications.

    You have now completed this lab.

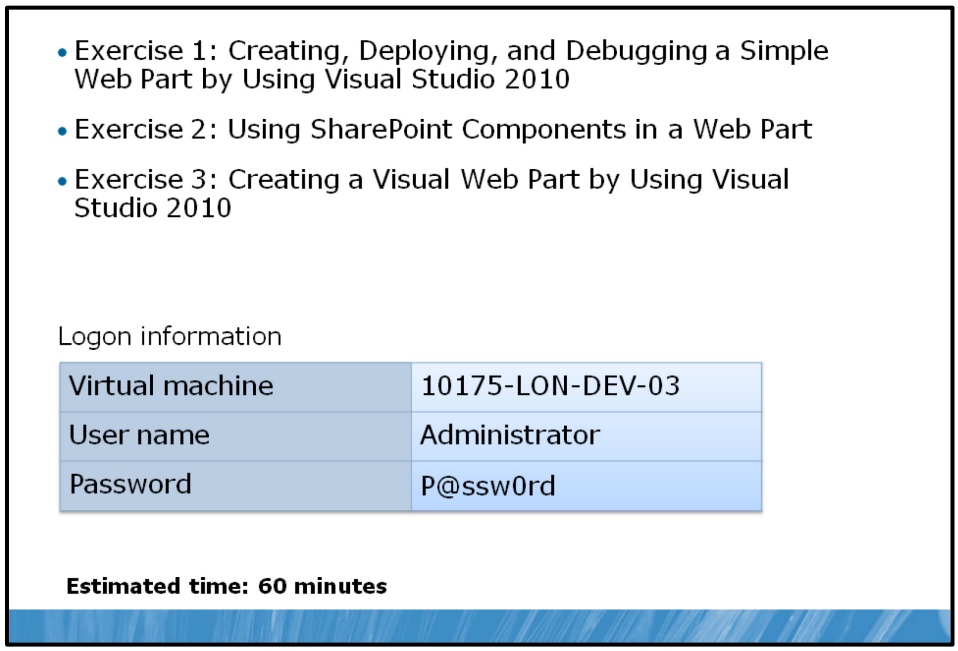
Results: After this exercise, you should have verified that all lists have been deployed and can be used.

Module 3

Lab Instructions: Developing SharePoint 2010 Web Parts

Contents:

Lab: Creating SharePoint 2010 Web Parts by Using Visual Studio 2010



Log On to the Virtual Machine for This Lab

For this lab, you will use the available virtual machine environment. Before you begin the lab, you must log on to the **10175-LON-DEV-03** virtual machine as **Administrator**, with a password of **P@ssw0rd**.

Exercise 1: Creating, Deploying, and Debugging a Simple Web Part by Using Visual Studio 2010

Scenario

In this exercise, you create a Web Part that renders literal text to start with. You then deploy and debug the Web Part by using Visual Studio tools.

Note: You will develop the Web Part into a more useful example in subsequent exercises.

The main tasks for this exercise are as follows:

1. Create an empty SharePoint project.
2. Add a standard Web Part to the project.
3. Debug the Web Part.

▶ Task 1: Create an empty SharePoint project

1. On the **Start** menu, click **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.
2. On the **File** menu, point to **New**, and then click **Project**.
3. In the **Installed Templates** section, expand the **Visual C#** node, expand the **SharePoint** node, and then click **2010**.
4. Click **Empty SharePoint Project.**
5. In the **Name** text box, type **Lab03**.
6. In the **Location** text box, type **E:\Student\Lab03\Starter.**
7. Click **OK.**
   The SharePoint Customization Wizard appears.
8. In the **What local site do you want to use for debugging?** text box, type **http://sharepoint**.
9. In the **What is the trust level for this SharePoint** solution section, click the Deploy as a farm solution option.
10. Click **Finish**.

    The project is created.

▶ Task 2: Add a standard Web Part to the project

1. In the Solution Explorer window, right-click **Lab03**, point to **Add**, and then click **New Item**.
2. Click **Web Part**.

Note: Do not click Visual Web Part.

3. In the **Name** text box, type **TaskView** and then click **Add**.
   A new Web Part is added to the project, and the class file for the Web Part appears.
4. Place the cursor between the opening and closing braces of the **CreateChildControls** method, and then press ENTER.
5. In the space you have created, type the following code:

```
LiteralControl myMessage =
new LiteralControl("<H5>People and Tasks</H5>");
this.Controls.Add(myMessage);
```

6. Click in the first line of code that you have just typed.
7. On the **Debug** menu, click **Toggle Breakpoint**.
8. On the **File** menu, click **Save All**.

Task 3: Debug the Web Part

1. In the Solution Explorer window, right-click **Lab03** and then click **Deploy**.
   Microsoft SharePoint builds, packages, and deploys the Web Part.
2. Start Windows® Internet Explorer®.
3. On the **Site Actions** menu, click **New Page**.
4. In the **New page name** text box, type **Recruitment** and then click **Create**.
5. On the ribbon, click the **Insert** tab.
6. On the ribbon, click **Web Part**.
7. In the **Categories** section, click **Custom**.
8. In the **Web Parts** section, click **TaskView**.
9. Click **Add**.
   The Web Part is added to the page.
10. On the ribbon, click **Save and Close**.
11. Close Internet Explorer.
12. Switch to Microsoft Visual Studio.
13. Press F5.
    SharePoint builds, packages, and deploys the Web Part, and then attaches the debugger to the w3p3.exe process.
14. If the **Debugging Not Enabled** dialog box appears, click **Modify the Web.config file to enable debugging** option, and then click **OK**.
    Internet Explorer starts.
15. On the **Quick Launch** bar, click **Site Pages**.
16. Click **Recruitment**.
    Visual Studio becomes active, with code execution paused on the breakpoint you added in the previous task.
17. In Visual Studio, on the **Debug** toolbar, click **Step Into**.

    If a message box appears informing you that the web server process that was being debugged has been terminated by Internet Information Services, click **OK**; you will then need to stop the debugger, close Internet Explorer, and start from step 13 again. If no message box appears, continue with step 18.

18. On the **Debug** toolbar, click **Continue**.
    Internet Explorer displays the home page with your Web Part.
19. Close Internet Explorer.
    The debugger in Visual Studio detaches from the w3wp.exe process and execution of the Web Part project ends.

Results: After this exercise, you should have deployed a Web Part and used the Visual Studio debugging tools to step through its code.

Exercise 2: Using SharePoint Components in a Web Part

Scenario

In this exercise, you add SharePoint components, such as the SharePoint DateTimeControl and the ListViewByQuery objects, to your Web Part. You also use the SPQuery class to tie these components together.

The main tasks for this exercise are as follows:

1. Create class-level variables for the Web Part.
2. Develop the CreateChildControls method to use SharePoint components.
3. Add event handler code for SharePoint components.
4. Test the Web Part.

▶ Task 1: Create class-level variables for the Web Part

1. Place the cursor at the beginning of the line of that reads:

   protected override void CreateChildControls and then press ENTER.
2. In the space you have just created, add the following code:

```
DateTimeControl filterDate;
ListViewByQuery MyCustomView;
SPQuery query;
```

3. On the line *above* the namespace declaration, add the following statement:

```
using Microsoft.SharePoint.Utilities;
```

4. On the **File** menu, click **Save All**.

▶ Task 2: Develop the CreateChildControls method to use SharePoint components

1. Place the cursor at the end of the last line of code that you previously added to the **CreateChildControls** method, and then press ENTER.
2. In the space that you have just created, add the following code:

```
filterDate = new DateTimeControl();
filterDate.DateOnly = true;
filterDate.AutoPostBack = true;
```

3. Press ENTER to create a new line.
4. Type **filterDate.DateChanged +=**.
5. Press TAB.
6. Press TAB.

   Visual Studio enters the event handler for you.
7. Place the cursor at the end of the last line of code that you previously added to the **CreateChildControls** method, and then press ENTER.
8. In the space that you have just created, add the following code:

```
SPWeb thisWeb = SPContext.Current.Web;
MyCustomView = new ListViewByQuery();
MyCustomView.List = thisWeb.Lists["Interviews"];
query = new SPQuery(MyCustomView.List.DefaultView);
query.ViewFields = "<FieldRef Name='Title' />"
  + "<FieldRef Name='AssignedTo' /><FieldRef Name='DueDate' />";
MyCustomView.Query = query;
```

```
LiteralControl filterMessage
  = new LiteralControl("Tasks due on or before:");
this.Controls.Add(filterMessage);
this.Controls.Add(new LiteralControl("<br />"));
this.Controls.Add(filterDate);
this.Controls.Add(new LiteralControl("<br />"));
this.Controls.Add(MyCustomView);
```

▶ Task 3: Add event handler code for SharePoint components

1. Delete the existing line of code in the **filterDate_DateChanged** event handler.
2. Add the following code to the **filterDate_DateChanged** event handler:

```
string camlQuery = "<Where><Leq><FieldRef Name='DueDate' />"
  + "<Value Type='DateTime'>"
  + SPUtility.CreateISO8601DateTimeFromSystemDateTime
  (filterDate.SelectedDate)
  + "</Value></Leq></Where>";
query.Query = camlQuery;
MyCustomView.Query = query;
```

3. On the **File** menu, click **Save All**.

▶ Task 4: Test the Web Part

1. In the Solution Explorer window, right-click **Lab03** and then click **Deploy**.

   SharePoint builds, packages, and deploys the Web Part.
2. On the **Start** menu, click **Internet Explorer**.
3. On the **Quick Launch** bar, click **Site Pages**.
4. Click **Recruitment**.

   The page shows your updated Web Part.
5. On the ribbon, click **Navigate Up**, and then click **HR Intranet**.
6. On the **Quick Launch** bar, click **Site Pages**.
7. Click **Recruitment**.

   Your Web Part currently shows all tasks in the Interviews list.
8. In your Web Part, click the date picker control.
9. Click the date that represents 14 May, 2011.

   Your Web Part currently shows the task in the Interviews list for interviewing Svetlana.
10. In your Web Part, click the date picker control.
11. Click the date that represents 19 June, 2011.

    Your Web Part currently shows the tasks in the Interviews list for interviewing Svetlana and Ivo.
12. Close Internet Explorer.

Results: After this exercise, you should have created and deployed a Web Part that uses SharePoint components.

Exercise 3: Creating a Visual Web Part by Using Visual Studio 2010

Scenario

In this exercise, you create a Visual Web Part, design its graphical user interface by adding a TreeView control, and add code to the Visual Web Part to display candidate and recruitment data from SharePoint lists in the TreeView control. You then deploy and test the Visual Web Part in the Recruitment page of the HR Intranet site.

The main tasks for this exercise are as follows:

1. Add a Visual Web Part to a Visual Studio project and design its graphical user interface.
2. Develop the code for the Visual Web Part.
3. Deploy and test the Visual Web Part.

▶ Task 1: Add a Visual Web Part to the project and design its graphical user interface

1. In the Solution Explorer window, right-click **Lab03**, point to **Add**, and then click **New Item**.
2. Click **Visual Web Part**.

> **Note:** Do not click Web Part.

3. In the **Name** text box, type **Overview** and then click **Add**.

   A new Visual Web Part is added to the project, and the OverviewUserControl.ascx markup file for the user control in the Visual Web Part appears.

4. Add the following markup to the end of the **OverviewUserControl.ascx** file:

```
<asp:TreeView ID="overviewTree"
  runat="server"
  ShowLines="true"
  EnableViewState="true">
</asp:TreeView>
```

5. On the **File** menu, click **Save All**.

▶ Task 2: Develop the code for the Visual Web Part

1. On the **View** menu, click **Code**.
2. On the line *above* the namespace declaration, add the following statement:

```
using Microsoft.SharePoint;
```

3. Add the following code between the start and end braces of the **Page_Load** method:

```
try
{
  overviewTree.Nodes.Clear();
  SPWeb thisWeb = SPContext.Current.Web;
  SPList candidates = thisWeb.Lists["Outcomes"];
  SPQuery itemMashup = new SPQuery();
  itemMashup.Joins =
    "<Join Type='LEFT' ListAlias='Candidates'><Eq>"
    + "<FieldRef Name='Candidate' RefType='Id' />"
    + "<FieldRef List='Candidates' Name='ID' /></Eq>"
    + "</Join>";
  itemMashup.ProjectedFields =
    "<Field Name='Applicant' Type='Lookup' List='Candidates'"
```

```
      + " ShowField='Title' />"
    + "<Field Name='HomeCity' Type='Lookup' List='Candidates'"
      + " ShowField='HomeCity' />";
  itemMashup.ViewFields = "<FieldRef Name='Applicant' />"
    + "<FieldRef Name='HomeCity' />"
    + "<FieldRef Name='Title' />"
    + "<FieldRef Name='Interviewer' />"
    + "<FieldRef Name='Offer' />";
  SPListItemCollection allCandidates =
    candidates.GetItems(itemMashup);
  foreach (SPListItem item in allCandidates)
  {
   TreeNode applicant = new TreeNode(item["Applicant"].ToString(),
     null,null,thisWeb.Lists["Candidates"].DefaultViewUrl,
     "_self");
   TreeNode opportunity = new TreeNode(item["Title"].ToString(),
    null, null, thisWeb.Lists["Outcomes"].DefaultViewUrl,
     "_self");
   TreeNode homeCity = new TreeNode(item["HomeCity"].ToString(),
     null, null, thisWeb.Lists["Candidates"].DefaultViewUrl,
     "_self");
   TreeNode interviewer = new TreeNode("Interviewed by: " +
     item["Interviewer"].ToString(), null, null,
     thisWeb.Lists["Interviews"].DefaultViewUrl, "_self");
   TreeNode offered =
     new TreeNode(bool.Parse(item["Offer"].ToString()) == true ?
     "Job Offered" : "Rejected", null, null,
     thisWeb.Lists["Outcomes"].DefaultViewUrl, "_self");
   applicant.ChildNodes.Add(opportunity);
   applicant.ChildNodes.Add(homeCity);
   applicant.ChildNodes.Add(interviewer);
   applicant.ChildNodes.Add(offered);
   overviewTree.Nodes.Add(applicant);
  }
  overviewTree.ExpandAll();
}
catch (Exception ex)
{
  overviewTree.Nodes.Add(new TreeNode("Err" + ex.Message));
}
```

4. On the **File** menu, click **Save All**.

Task 3: Deploy and test the Visual Web Part

1. In the Solution Explorer window, right-click **Lab03** and then click **Deploy**.

   SharePoint builds, packages, and deploys the Web Part.
2. On the **Start** menu, click **Internet Explorer**.
3. On the **Quick Launch** bar, click **Site Pages**.
4. Click **Recruitment**.
5. On the ribbon, click **Edit**.
6. On the ribbon, click the **Insert** tab.
7. On the ribbon, click **Web Part**.
8. In the **Categories** section, click **Custom**.
9. In the **Web Parts** section, click **Overview**.
10. Click **Add**.

    The Web Part is added to the page.

11. On the ribbon, click **Save and Close**.
12. Expand all the nodes in the tree view control to view the data from the lists in HR Intranet site.

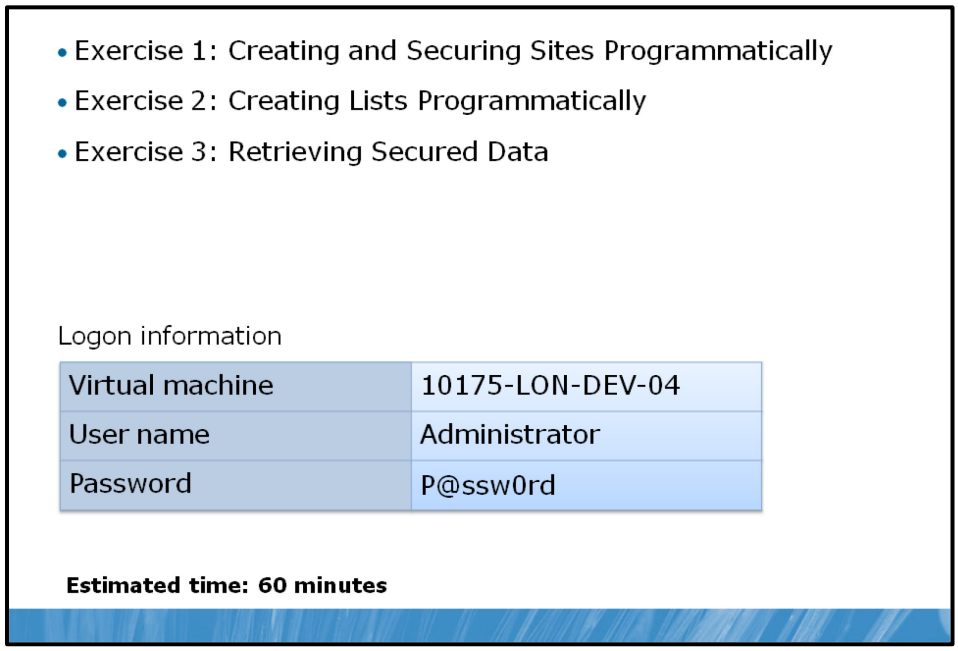
Results: After this exercise, you should have developed and deployed a Visual Web Part.

Module 4

Lab Instructions: Working with SharePoint Objects on the Server

Contents:

Lab: Creating and Manipulating Server-Side Objects



Log On to the Virtual Machine for This Lab

For this lab, you use the available virtual machine environment. Before you begin the lab, you must log on to the **10175-LON-DEV-04** virtual machine as **Administrator** with the password **P@ssw0rd**.

Exercise 1: Creating and Securing Sites Programmatically

Scenario

In this exercise, you create a secured subsite that only the HR Managers (KrishnaS and MartinR) can access. Later in this lab, you also create a list of job definitions, including salary range details, which is why the site must be secured for only the HR Managers to access.

The main tasks for this exercise are as follows:

1. Creating a new SharePoint project.
2. Adding an application page for creating and securing sites.

▶ Task 1: Create an empty SharePoint project

1. On the **Start** menu, click **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.
2. On the **File** menu, point to **New**, and then click **Project**.
3. In the **Installed Templates** section, expand the **Visual C#** node, expand the **SharePoint** node, and then click **2010**.
4. Click **Empty SharePoint Project**.
5. In the **Name** text box, type **Lab04**.
6. In the **Location** text box, type **E:\Student\Lab04\Starter**. Click **OK**.

   The SharePoint Customization Wizard appears.
7. In the **What local site do you want to use for debugging?** text box, type **http://sharepoint**.
8. In the **What is the trust level for this SharePoint solution** section, click the **Deploy as a farm solution** option.
9. Click **Finish**.

   The project is created.

▶ Task 2: Add an application page for creating and securing sites

1. In the Solution Explorer window, right-click **Lab04**, point to **Add**, and then click **New Item**.
2. Click **Application Page**.
3. In the **Name** text box, type **CreateJobDef.aspx**, and then click **Add**.
4. Add the following markup between the opening and closing tags of the **<asp:Content>** element that has an **ID** of **Main**:

```
<asp:Button ID="creator" runat="server"
  Text="Create Job Definition List"/>
<br />
<asp:Label ID="status" runat="server" Text=""></asp:Label>
```

5. Change the markup between the opening and closing tags of the **<asp:Content>** element that has an **ID** of **PageTitle** to the following:

```
Job Definitions
```

6. Change the markup between the opening and closing tags of the **<asp:Content>** element that has an **ID** of **PageTitleInTitleArea** to the following:

```
Utility for creating Job Definitions
```

7. On the **View** menu, click **Code**.
8. Place the cursor between the opening and closing braces of the **Page_Load** function, and then press ENTER.
9. On the new line that you have just created, type **creator.Click +=**.
10. Press TAB.
11. Press TAB.

    Microsoft Visual Studio enters the event handler for you.
12. After the closing brace of the **creator_Click** function, add the following function:

```
void createList(SPWeb subWeb)
{

}
```

Note: You develop the **createList** function later in this lab.

13. Replace the existing code in the **creator_Click** function with the following:

```
SPWeb thisWeb = SPContext.Current.Web;
SPWeb newWeb = null;
if (!thisWeb.Webs["JobData"].Exists)
{
  try
  {
    newWeb = thisWeb.Webs.Add("JobData",
      "Job Data",
      "Data for Jobs",
      1033,
      "STS#1",
      true,
      false);
    SPRoleAssignment roleAssign =
      new SPRoleAssignment(@"SHAREPOINT\KrishnaS",
      "krishnas@sharepoint.com", @"SHAREPOINT\KrishnaS",
      "HR Manager");
    SPRoleDefinition roleDef =
      newWeb.RoleDefinitions["Contribute"];
    roleAssign.RoleDefinitionBindings.Add(roleDef);
    newWeb.RoleAssignments.Add(roleAssign);
    newWeb.Update();
    roleAssign = new SPRoleAssignment(@"SHAREPOINT\MartinR",
      "martinr@sharepoint.com", @"SHAREPOINT\MartinR",
      "HR Manager");
    roleDef = newWeb.RoleDefinitions["Contribute"];
    roleAssign.RoleDefinitionBindings.Add(roleDef);
    newWeb.RoleAssignments.Add(roleAssign);
    newWeb.Update();
    createList(newWeb);
    status.Text =
      "Job Data site and Job Definitions list have been added";
    creator.Enabled = false;
  }
  catch (Exception webEx)
  {
    status.Text = webEx.Message;
```

```
  }
  finally
  {
    newWeb.Dispose();
  }
  return;
}
```

14. After the code you have just typed, add the following code:

```
using (SPWeb existingWeb = thisWeb.Webs["JobData"])
{
  try
  {
    SPList jobDefToCreate = existingWeb.Lists.TryGetList
      ("Job Definitions");
    if(jobDefToCreate != null)
    {
      status.Text =
        "Job Data site and Job Definitions already exist";
      creator.Enabled = false;
      return;
    }
    createList(existingWeb);
    status.Text = "Job Data site already exists. "
      + "Job Definitions list have been added";
    creator.Enabled = false;
  }
  catch(Exception listEx)
  {
    status.Text = listEx.Message;
  }
}
```

15. On the **File** menu, click **Save All**.

Results: After this exercise, you should have written and tested code that creates and secures sites programmatically.

Exercise 2: Creating Lists Programmatically

Scenario

In this exercise, you develop code that creates the Job Definitions list in the secured subsite and adds items for some job definitions.

The main tasks for this exercise are as follows:

1. Completing the code for the **createList** function.
2. Deploying and testing the application page.

▶ Task 1: Complete the code for the **createList** function

1. Add the following code to the **createList** function:

```
Guid jobDefGuid = subWeb.Lists.Add
  ("Job Definitions", "Jobs, Salary Ranges, and Descriptions",
  SPListTemplateType.GenericList);
SPList jobDef = subWeb.Lists[jobDefGuid];
jobDef.Fields.Add("MinSalary", SPFieldType.Currency, true);
jobDef.Fields.Add("MaxSalary", SPFieldType.Currency, true);
jobDef.Fields.Add("JobDescription", SPFieldType.Text, false);
jobDef.OnQuickLaunch = true;
jobDef.Update();

SPView vw = jobDef.Views["All Items"];
vw.ViewFields.Add("MinSalary");
vw.ViewFields.Add("MaxSalary");
vw.ViewFields.Add("JobDescription");
vw.InlineEdit = "TRUE";
vw.Update();

SPListItem newDef;
newDef = jobDef.Items.Add();
newDef["Title"]= "Developer";
newDef["MinSalary"] = "40000";
newDef["MaxSalary"] = "80000";
newDef["JobDescription"] = "SharePoint or other Web Developer";
newDef.Update();

newDef = jobDef.Items.Add();
newDef["Title"] = "Analyst";
newDef["MinSalary"] = "40000";
newDef["MaxSalary"] = "90000";
newDef["JobDescription"] = "Business Analyst";
newDef.Update();

newDef = jobDef.Items.Add();
newDef["Title"] = "Lead Developer";
newDef["MinSalary"] = "60000";
newDef["MaxSalary"] = "100000";
newDef["JobDescription"] = "Developer and Team Leader";
newDef.Update();

newDef = jobDef.Items.Add();
newDef["Title"] = "Solution Architect";
newDef["MinSalary"] = "80000";
newDef["MaxSalary"] = "120000";
newDef["JobDescription"] = "Experienced Solution Architect";
newDef.Update();
```

2. On the **File** menu, click **Save All**.

▶ Task 2: Deploy and test the application page

1. In the Solution Explorer window, right-click **Lab04**, and then click **Deploy**.
2. On the **Start** menu, click **Internet Explorer**.
3. In the **Address Bar**, type **http://sharepoint/_layouts/lab04 /CreateJobDef.aspx**, and press ENTER.
4. Click **Create JobDefinition List**.

   After a few seconds you will be notified that the site and list have been created.
5. On the **Quick Launch** bar, click **All Site Content**.
6. In the **Sites and Workspaces** section, click **Job Data**.
7. On the **Quick Launch** bar, click **Job Definitions**, and review the data.
8. On the **Site Actions** menu, click the **Site Permissions**.

   Note that the site has unique permissions, and note that KrishnaS and MartinR have been granted contribute permissions.
9. Close Internet Explorer.

Results: After this exercise, you should have written and tested code that create lists and list items.

Exercise 3: Retrieving Secured Data

Scenario

The HR Intranet site needs for all HR staff to see open positions for each of the job definitions that is stored in the secured subsite. Therefore, you create a pair of connected Web Parts.

One Web Part retrieves a list of job titles from the job definitions in the secured subsite and uses elevated permissions to get this data if necessary. Note that only the salary information in the secured subsite is considered sensitive, not the job titles, so this is a design decision that has already been made—it is OK to show these job titles to all users. This Web Part then acts as a provider in the Web Part connection scenario.

The other Web Part consumes the data provided by the first Web Part and uses that information to build LINQ to SharePoint queries to retrieve details of open positions from the unsecured area of the HR Intranet site.

The main tasks for this exercise are as follows:

1. Creating a pair of connected Web Parts.
2. Creating an interface for the connection between the Web Parts.
3. Implementing the provider Web Part to retrieve data from the secured subsite.
4. Generating LINQ entities for use in the consumer Web Part.
5. Implementing the consumer Web Part.
6. Deploying and testing the visual Web Part.

▶ Task 1: Create a pair of connected Web Parts

1. Switch to Visual Studio.
2. In the Solution Explorer window, right-click **Lab04**, point to **Add**, and then click **New Item**.
3. Click **Web Part**.
4. In the **Name** text box, type **JobDefinitions**, and then click **Add**.
5. In the Solution Explorer window, right-click **Lab04**, point to **Add**, and then click **New Item**.
6. Click **Web Part**.
7. In the **Name** text box, type **OpenPositions**, and then click **Add**.

▶ Task 2: Create an interface for the connection between the Web Parts

1. Click the **JobDefinitions.cs** tab.
2. Place the cursor immediately after the opening brace of the namespace **Lab04.JobDefinitions**, and then press ENTER.
3. In the space you have just created, type the following code:

```
public interface IJobDef
{
  string JobDef
  {
    get;
    set;
  }
}
```

Task 3: Implement the provider Web Part to retrieve data from the secured subsite

1. Edit the line of code that reads **public class JobDefinitions : WebPart** so that it reads as follows:

```
public class JobDefinitions : WebPart, IJobDef
```

2. Place the cursor immediately before the **CreateChildControls** function declaration, and then press ENTER.

3. In the space you have just created, type the following code:

```
private DropDownList allJobs;
private string _JobDef;
public string JobDef
{
  get
  {
    return (_JobDef);
  }
  set
  {
    _JobDef = value;
  }
}
```

4. After the closing brace of the **JobDef** property, add the following code:

```
[ConnectionProvider("Job")]
public IJobDef SendJobName()
{
    return (this);
}
```

5. Add the following code to the **CreateChildControls** function:

```
allJobs = new DropDownList();
allJobs.AutoPostBack = true;
allJobs.EnableViewState = true;
populateList();
```

   You create the **populateList** function later in this exercise.

6. After the code you have just typed, type **allJobs.SelectedIndexChanged +=**.

7. Press TAB.

8. Press TAB.

   Visual Studio enters the event handler for you.

9. Add the following code to the code you have already typed in the **CreateChildControls** function:

```
this.Controls.Add(new LiteralControl
  ("Job Definition Filter:<br/>"));
this.Controls.Add(allJobs);
```

10. Replace the existing code in the **allJobs_SelectedIndexChanged** event handler function with the following:

```
_JobDef = allJobs.SelectedValue;
```

11. Add the following function after the closing brace of the **CreateChildControls** function:

```
void populateList()
{
  SPWeb jobDefWeb=null;
  SPWeb thisWeb = SPContext.Current.Web;
  SPList jobDefList;
```

```
  thisWeb.Site.CatchAccessDeniedException = false;
  try
  {
    allJobs.Items.Add("Pick a Job");
    allJobs.Items.Add("All Jobs");
    jobDefWeb = SPContext.Current.Web.Webs["JobData"];
    jobDefList = jobDefWeb.Lists["Job Definitions"];
    foreach (SPListItem item in jobDefList.Items)
    {
      allJobs.Items.Add(item.Title);
    }
  }
  catch (UnauthorizedAccessException secEx)
  {
    SPUser privilegedAccount =
      SPContext.Current.Web.AllUsers[@"SHAREPOINT\SYSTEM"];
    SPUserToken privilegedToken = privilegedAccount.UserToken;
    using (SPSite elevatedSite =
      new SPSite(SPContext.Current.Web.Url, privilegedToken))
    {
      using (SPWeb elevatedWeb = elevatedSite.OpenWeb())
      {
        jobDefWeb = elevatedWeb.Webs["JobData"];
        jobDefList = jobDefWeb.Lists["Job Definitions"];
        foreach (SPListItem item in jobDefList.Items)
        {
          allJobs.Items.Add(item.Title);
        }
      }
    }
  }
  finally
  {
    thisWeb.Site.CatchAccessDeniedException = true;
    jobDefWeb.Dispose();
  }
}
```

12. On the **File** menu, click **Save All**.

▶ Task 4: Generate LINQ entities for use in the consumer Web Part

1. Click **Start**, and then click **Run**.
2. Type **cmd**, and then press ENTER.
3. At the command prompt, type **cd "C:\Program Files\Common Files \Microsoft Shared\Web Server Extensions\14\bin"**, and press ENTER.
4. At the command prompt, type **SPMetal /web:http://sharepoint /code:E:\Student\HREntities.cs /language:csharp**, and press ENTER.

   You are notified that content types for list form templates were excluded. This is expected behavior.
5. Close the command prompt.

▶ Task 5: Implement the consumer Web Part

1. Switch to Visual Studio.
2. In the Solution Explorer window, right-click **Lab04**, and then click **Add Reference**.
3. Click the **Browse** tab, and then browse to the **C:\Program Files\Common Files \Microsoft Shared\Web Server Extensions\14\ISAPI** folder.

4. Click **Microsoft.SharePoint.Linq.dll** and then click **OK**.
5. In the Solution Explorer window, right-click **Lab04**, point to **Add**, and then click **Existing Item**.
6. Browse to the **E: Student** folder.
7. Click **HREntities.cs**, and then click **Add**.
8. Click the **OpenPositions** tab.
9. In the space immediately above the namespace **Lab04.OpenPositions** declaration, add the following **using** statements:

```
using System.Linq;
using Microsoft.SharePoint.Linq;
```

10. Place the cursor immediately before the **CreateChildControls** function declaration, and then press ENTER.
11. In the space you have just created, type the following code:

```
private string jobDefinition = "All Jobs";
[ConnectionConsumer("Job")]
public void GetJob(Lab04.JobDefinitions.IJobDef Job)
{
  if (Job != null)
  {
    jobDefinition = Job.JobDef;
  }
}
```

12. Add the following code to the **CreateChildControls** function:

```
Table jobTable = new Table();
TableRow jobRow = new TableRow();
TableCell jobDetail = new TableCell();
jobDetail.Text = "Job";
jobDetail.Font.Bold = true;
jobDetail.HorizontalAlign = HorizontalAlign.Center;
jobRow.Cells.Add(jobDetail);

jobDetail = new TableCell();
jobDetail.Text = "Location";
jobDetail.Font.Bold = true;
jobDetail.HorizontalAlign = HorizontalAlign.Center;
jobRow.Cells.Add(jobDetail);

jobDetail = new TableCell();
jobDetail.Text = "Interviewer";
jobDetail.Font.Bold = true;
jobDetail.HorizontalAlign = HorizontalAlign.Center;
jobRow.Cells.Add(jobDetail);

jobTable.Rows.Add(jobRow);

this.Controls.Add(new LiteralControl("Open Positions: " +
  jobDefinition + "<br />"));
HREntitiesDataContext hrDC = new
  HREntitiesDataContext("http://sharepoint");
if (jobDefinition != "All Jobs")
{
  var filteredJobs = from vacancyList in hrDC.CurrentVacancies
    where(vacancyList.Title.Equals(jobDefinition))
    orderby vacancyList.Title
    select new { vacancyList.Title,
```

```
      vacancyList.Location, vacancyList.InterviewerImnName };
  foreach (var job in filteredJobs)
  {
    jobRow = new TableRow();
    jobDetail = new TableCell();
    jobDetail.Text = job.Title;
    jobRow.Cells.Add(jobDetail);

    jobDetail = new TableCell();
    jobDetail.Text = job.Location;
    jobRow.Cells.Add(jobDetail);

    jobDetail = new TableCell();
    jobDetail.Text = job.InterviewerImnName;
    jobRow.Cells.Add(jobDetail);

    jobTable.Rows.Add(jobRow);
  }
  this.Controls.Add(jobTable);
  return;
}
var unfilteredJobs = from vacancyList in hrDC.CurrentVacancies
  where (vacancyList.Title.Equals(jobDefinition))
  orderby vacancyList.Title
  select new { vacancyList.Title, vacancyList.Location,
    vacancyList.InterviewerImnName };
foreach (var job in unfilteredJobs)
{
  jobRow = new TableRow();
  jobDetail = new TableCell();
  jobDetail.Text = job.Title;
  jobRow.Cells.Add(jobDetail);

  jobDetail = new TableCell();
  jobDetail.Text = job.Location;
  jobRow.Cells.Add(jobDetail);

  jobDetail = new TableCell();
  jobDetail.Text = job.InterviewerImnName;
  jobRow.Cells.Add(jobDetail);

  jobTable.Rows.Add(jobRow);
}
this.Controls.Add(jobTable);
```

13. On the **File** menu, click **Save All**.

▶ Task 6: Deploy and test the visual Web Part

1. In the Solution Explorer window, right-click **Lab04**, and then click **Deploy**.

   Microsoft SharePoint builds, packages, and deploys the Web Parts.

2. On the **Start** menu, click **Internet Explorer**.
3. On the **Quick Launch** bar, click **Site Pages**.
4. Click **Recruitment**.
5. On the ribbon, click **Edit**.
6. On the ribbon, click the **Insert** tab.
7. On the ribbon, click **Web Part**.

8. In the **Categories** section, click **Custom**.
9. In the **Web Parts** section, click **JobDefinitions**.
10. Click **Add**.

    The Web Part is added to the page.
11. On the ribbon, click **Web Part**.
12. In the **Categories** section, click **Custom**.
13. In the **Web Parts** section, click **OpenPositions**.
14. Click **Add**.

    The Web Part is added to the page.
15. Click the **JobDefinitions Web Part** menu for the **JobDefinitions** Web Part.
16. Click **Edit Web Part**.
17. Click the **JobDefinitions Web Part** menu for the **JobDefinitions** Web Part.
18. Click **Connections**, click **Send Job To**, and then click **Open Positions**.
19. On the ribbon, click **Save and Close**.
20. In the **Job Definition Filter** drop-down list, click **Developer**.

    Developer jobs are displayed in the OpenPositions Web Part.
21. In the **Job Definition Filter** drop-down list, click **All Jobs**.

    All jobs are displayed in the OpenPositions Web Part.
22. Close all applications. You have now completed this lab.

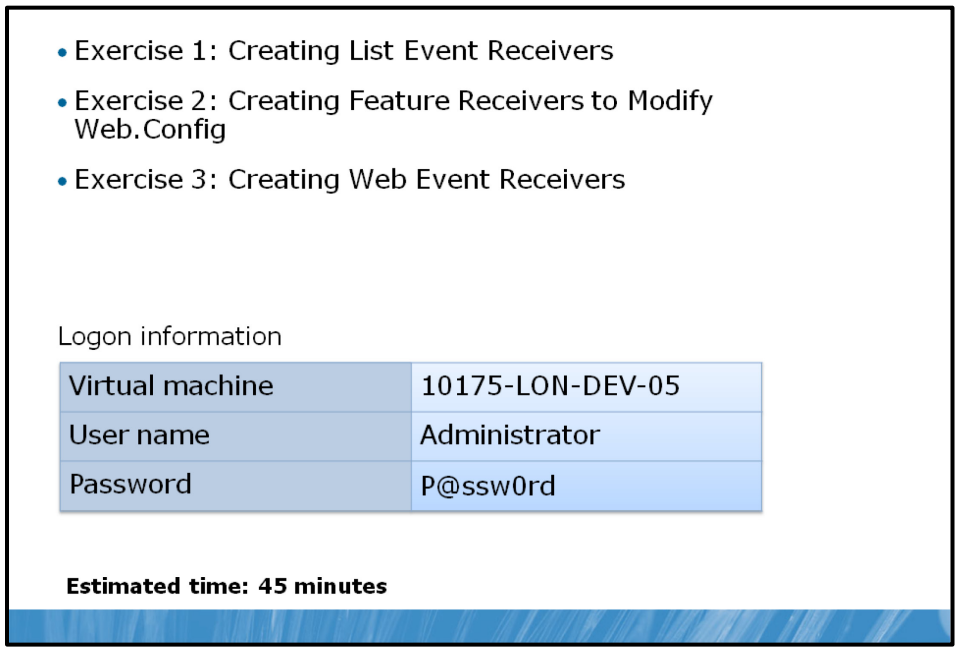
Results: After this exercise, you should have created and tested connected Web Parts that render data from specific lists in the HR Intranet site.

Module 5

Lab Instructions: Creating Event Receivers and Application Settings

Contents:

Lab: Creating Event Receivers and Web.Config Modifications



Log On to the Virtual Machine for This Lab

For this lab, use the available virtual machine environment. Before you begin the lab, log on to the **10175-LON-DEV-05** virtual machine as **Administrator**, with a password of **P@ssw0rd**.

Exercise 1: Creating List Event Receivers

Scenario

The HR Intranet contains a secured sub-site for storing job definitions and salary ranges. Only HR Managers can view this data directly. However, other users who are not HR Managers can add items to the Outcomes list to keep track of the results of interviews.

Part of this process is that the user adds a job title for the interview outcome. The HR managers want the other users to be able to add these interview outcome items, but they also want to restrict the values added to the Title of each item to those jobs that are defined in the secure sub-site. Because the security would not allow you to provide a lookup column to this secured data (or at least would not allow a non-manager to use the lookup), you create list event receivers for validating the data added to the interview outcome list.

In essence, you create event handlers for the ItemAdding event and the ItemUpdating event, and you cancel the events if the values in the Title field do not match job definitions from the secured subsite.

The main tasks for this exercise are as follows:

1. Create an empty SharePoint project.
2. Add event receivers to the InterviewOutcomes list.
3. Develop the event handlers in the event receiver class.
4. Deploy and test the list event receiver.

▶ Task 1: Create an empty SharePoint project

1. On the **Start** menu, click **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.
2. On the **File** menu, point to **New**, and then click **Project**.
3. In the **Installed Templates** section, expand the **Visual C#** node, expand the **SharePoint** node and then click **2010**.
4. Click **Empty SharePoint Project**.
5. In the **Name** text box, type **Lab05**.
6. In the **Location** text box, type **E:\Student\Lab05\Starter**.
7. Click **OK**.

   The Microsoft SharePoint Customization Wizard appears.
8. In the **What local site do you want to use for debugging?** text box, type **http://sharepoint**.
9. In the **What is the trust level for this SharePoint** solution section, click the **Deploy as a farm solution** option.
10. Click **Finish**.

    The project is created.

▶ Task 2: Add event receivers to the InterviewOutcomes list

1. In the Solution Explorer window, right-click **Lab05**, point to **Add** and then click **New Item**.
2. Click **Event Receiver**.

3. In the **Name** text box, type **CheckJobs** and then click **Add**.

   The SharePoint Customization wizard appears.

4. In the **What type of event receiver do you want?** list, click **List Item Events**.
5. In the **What item should be the event source?** lists, click **InterviewOutcomes**.
6. In the **Handle the following events** checkbox list, check the **An item is being added** checkbox, and the **An item is being updated** checkbox.
7. Click **Finish**.

   Microsoft Visual Studio adds an event receiver class with stub methods for the events you chose to handle.

▶ Task 3: Develop the event handlers in the event receiver class

1. Add the following function after the closing brace of the **ItemUpdating** event handler:

```
bool checkItem(SPItemEventProperties properties)
{
  string jobTitle =
    properties.AfterProperties["Title"].ToString();
  bool allowed = false;
  SPWeb jobDefWeb = null;
  SPList jobDefList;
  SPUser privilegedAccount =
    properties.Web.AllUsers[@"SHAREPOINT\SYSTEM"];
  SPUserToken privilegedToken = privilegedAccount.UserToken;
  try
  {
    using (SPSite elevatedSite =
      new SPSite(properties.Web.Url, privilegedToken))
    {
      using (SPWeb elevatedWeb = elevatedSite.OpenWeb())
      {
        jobDefWeb = elevatedWeb.Webs["JobData"];
        jobDefList = jobDefWeb.Lists["Job Definitions"];
        foreach (SPListItem item in jobDefList.Items)
        {
          if (item["Title"].ToString() == jobTitle)
          {
            allowed = true;
            break;
          }
        }
      }
    }
    return (allowed);
  }
  finally
  {
    jobDefWeb.Dispose();
  }
}
```

2. Replace the existing code in the **ItemAdding** event handler with the following code:

```
try
{
  bool allowed = checkItem( properties);
  if (!allowed)
  {
    properties.Status = SPEventReceiverStatus.CancelWithError;
```

```
    properties.ErrorMessage =
     "The job is not defined in the Job Definitions List";
    properties.Cancel = true;
  }
}
catch (Exception ex)
{
  properties.Status = SPEventReceiverStatus.CancelWithError;
  properties.ErrorMessage = ex.Message;
  properties.Cancel = true;
}
```

3. Replace the existing code in the **ItemUpdating** event handler with the following code:

```
try
{
  bool allowed = checkItem(properties);
  if (!allowed)
  {
    properties.Status = SPEventReceiverStatus.CancelWithError;
    properties.ErrorMessage =
      "The job is not defined in the Job Definitions List";
      properties.Cancel = true;
  }
}
catch (Exception ex)
{
  properties.Status = SPEventReceiverStatus.CancelWithError;
  properties.ErrorMessage = ex.Message;
  properties.Cancel = true;
}
```

4. On the **File** menu, click **Save All**.

▶ Task 4: Deploy and test the list event receiver

1. In the Solution Explorer window, right-click **Lab05** and then click **Deploy**.
2. Use Windows® Internet Explorer® to browse to **http://sharepoint**.
3. On the **Quick Launch** bar, click **Outcomes**.
4. On the ribbon, click the **Items** tab.
5. On the ribbon, click **New Item**.
6. In the **Title** text box, type **Analyst/Developer**.
7. Click **Save**.

   An error dialog informs you that the job you entered in not defined in the Job Definitions list.
8. Close the dialog.
9. On the ribbon, click **New Item**.
10. In the **Title** text box, type **Developer**.
11. Click **Save**.

    The item is added to the list successfully.

Results: After this exercise, you should have developed, deployed, and tested an event receiver for list items.

Exercise 2: Creating Feature Receivers to Modify Web.Config

Scenario

In this exercise, you create a feature that an administrator can activate or deactivate in the HR Intranet. The feature includes feature receivers that run when the feature is activated or deactivated. The event handler that runs when the feature is activated adds an entry to the Web.config file. The event handler that runs when the feature is deactivated modifies that same entry in the Web.config file. The Web.config entry is used later in this lab to control whether sub-sites can be created in the HR Intranet.

The main tasks for this exercise are as follows:

1. Add a new feature and a feature receiver to the project.
2. Add code for modifying Web.config.

▶ Task 1: Add a new feature and feature receiver to the project

1. In the Solution Explorer window, right-click **Features** and then click **Add Feature**.

   Microsoft Visual Studio adds a new feature called Feature2.
2. In the Solution Explorer window, right-click **Feature2** and then click **Rename**.
3. Type **ControlProliferation** and press ENTER.
4. In the Solution Explorer window, double-click **ControlProliferation**.

   Visual Studio opens the Feature designer.
5. In the **Title** text box, type **Control Proliferation of Subsites**.
6. In the **Scope** drop-down list, click **Web**.
7. On the **File** menu, click **Save All**.
8. In the Solution Explorer window, right-click **ControlProliferation** and then click **Add Event Receiver**.

   Visual Studio adds a class to the project and opens it.
9. Select the three currently-commented out lines of code for the **FeatureActivated** event handler.
10. On the toolbar, click **Uncomment the selected lines**.
11. Select the three currently-commented out lines of code for the **FeatureDeactivating** event handler.
12. On the toolbar, click **Uncomment the selected lines**.

▶ Task 2: Add code for modifying the Web.config file

1. Add the following statement to the empty line above the **namespace** declaration:

```
using Microsoft.SharePoint.Administration;
```

2. Add the following function after the closing brace of the **FeatureDeactivating** event handler:

```
void setProliferationFlag(bool status)
{
  SPWebApplication webApp =
    SPWebApplication.Lookup(new Uri("http://SharePoint"));
  try
{
  SPWebConfigModification mySetting = null;
  if (status)
  {
    mySetting = new SPWebConfigModification();
```

```
    mySetting.Path = "configuration/appSettings";
    mySetting.Name =
      "add [@key='preventProliferation'] [@value='1']";
    mySetting.Sequence = 0;
    mySetting.Owner = "Lab05Owner";
    mySetting.Type =
      SPWebConfigModification.SPWebConfigModificationType
       .EnsureChildNode;
    mySetting.Value =
      "<add key='preventProliferation' value='1' />";
    webApp.WebConfigModifications.Add(mySetting);
  }
  else
  {
    foreach (SPWebConfigModification modification in
      webApp.WebConfigModifications)
    {
      if (modification.Owner == "Lab05Owner")
      {
        modification.Value =
          "<add key='preventProliferation' value='0' />";
      }
    }
  }
  webApp.Update();
  webApp.Farm.Services.GetValue<SPWebService>()
    .ApplyWebConfigModifications();
  }
  catch
  {
  }
}
```

3. Add the following code between the opening and closing braces of the **FeatureActivated** event handler:

```
setProliferationFlag(true);
```

4. Add the following code between the opening and closing braces of the **FeatureDeactivating** event handler:

```
setProliferationFlag(false);
```

5. On the **File** menu, click **Save All**.

Results: After this exercise, you should have added a Feature to your project, and added feature receivers to modify Web.config when the Feature is activated and deactivated.

Exercise 3: Creating Web Event Receivers

Scenario

In this exercise, you add a Web event receiver for the **WebAdding** event. This event receiver will use the **ConfigurationManager** class to read the **appSetting** value which you created in the previous exercise. Based on the value of this setting, the Web event receiver either allows or disallows the creation of sub-sites.

The main tasks for this exercise are as follows:

1. Add a Web event receiver.
2. Develop the Web event receiver to read the Web.config file.
3. Deploy and test the Web event receiver.

▶ Task 1: Add a Web event receiver

1. In the Solution Explorer window, right-click **Lab05**, point to **Add** and then click **New Item**.
2. Click **Event Receiver**.
3. In the **Name** text box, type **ControlSubsites**, and then click **Add**.

   The SharePoint Customization Wizard appears.
4. In the **What type of event receiver do you want?** list, click **Web Events**.
5. In the **Handle the following events** checkbox list, check the **A site is being provisioned** checkbox.
6. Click **Finish**.

   Visual Studio adds an event receiver class with stub methods for the event you chose to handle.

▶ Task 2: Develop the Web event receiver to read the Web.config file

1. Add the following statements to the empty line above the **namespace** declaration:

```
using System.Configuration;
using System.Web;
```

2. Replace the existing code in the **WebAdding** event handler with the following code:

```
string preventProliferation = ConfigurationManager
  .AppSettings["preventProliferation"].ToString();
if (preventProliferation == "1")
{
  properties.ErrorMessage = "Sub webs are not allowed";
  properties.Status = SPEventReceiverStatus.CancelWithError;
  properties.Cancel = true;
  return;
}
base.WebAdding(properties);
```

3. On the **File** menu, click **Save All**.

▶ Task 3: Deploy and test the Web event receiver

1. In the Solution Explorer window, right-click **Lab05** and then click **Deploy**.
2. Use Internet Explorer to browse to **http://sharepoint**.
3. On the **Site Actions** menu, click **Site Settings**.
4. In the **Site Actions** section of the page, click **Manage site features**.

Your feature called Control Proliferation of Subsites is included in the feature list.

5. If the Control Proliferation of Subsites feature is not currently active, click **Activate**.
6. On the **Site Actions** menu, click **New Site**.
7. Click **Team Site**.
8. In the **Title** text box, type **Test**.
9. In the **URL name** text box, type **Test**.
10. Click **Create**.

    You are notified that subwebs are not allowed.
11. Close the **Error** dialog and close the **Create** dialog.
12. Use Internet Explorer to browse to **http://sharepoint**.
13. On the **Site Actions** menu, click **Site Settings**.
14. In the **Site Actions** section of the page, click **Manage site features**.
15. Click **Deactivate** for the **Control Proliferation of Subsites** feature.
16. Click **Deactivate this feature**.
17. On the **Site Actions** menu, click **New Site**.
18. Click **Team Site**.
19. In the **Title** text box, type **Test**.
20. In the **URL name** text box, type **Test**.
21. Click **Create**.

    Your site is created successfully.
22. Close all applications.

    You have now completed this lab.

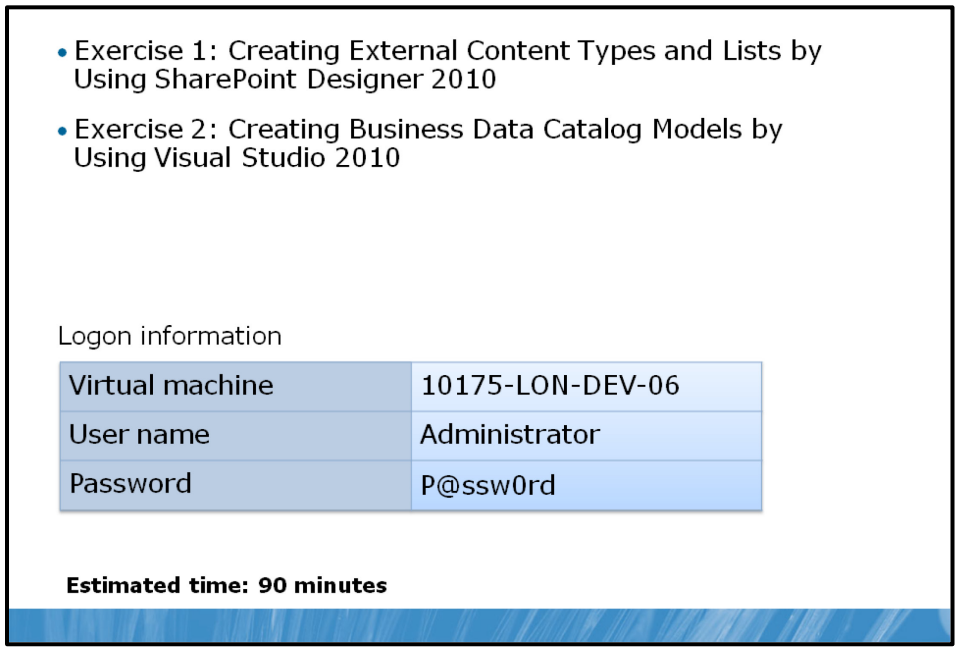
Results: After this exercise, you should have created a Web event receiver that reads from the Web.config file to determine whether to cancel WebAdding events.

Module 6

Lab Instructions: Developing Solutions by Using Business Connectivity Services

Contents:

Lab: Building Business Connectivity Services Solutions



Log On to the Virtual Machine for This Lab

For this lab, you use the available virtual machine environment. Before you begin the lab, you must log on to the **10175-LON-DEV-06** virtual machine as **Administrator** with the password **P@ssw0rd**.

Exercise 1: Creating External Content Types and Lists by Using SharePoint Designer 2010

Scenario

In this exercise, you use Business Connectivity Services to provide access to data stored in an external SQL Server database.

The main tasks for this exercise are as follows:

1. Review the design of the HRTrainingManagement database.
2. Create database connections.
3. Create external content types.
4. Create lists and forms for external data.
5. Secure and test access to external data in a SharePoint site.

▶ Task 1: Explore the HRTrainingManagement database

1. On the **Start** menu, click **All Programs**, click **Microsoft SQL Server 2008**, and then click **SQL Server Management Studio**.

   The Connect to Server dialog box appears.
2. In the **Server type** drop-down list, click **Database Engine**.
3. In the **Server name** drop-down list, type **SHAREPOINT\SharePoint**.
4. In the **Authentication** drop-down list, click **Windows Authentication**.
5. Click **Connect**.
6. In the Object Explorer window, expand **Databases**.
7. In the Object Explorer window, expand **HR Training Management**.
8. In the Object Explorer window, expand **Database Diagrams**.
9. In the Object Explorer window, double-click **dbo. Schema**.
10. Review the database design. Note that the **Training Supplier** table is not related to any other table. Also note that the **Training Objects** table is related to the **Student** table with a many-to-many relationship implemented by joins to the **Training Event** table.

▶ Task 2: Create a connection by using SharePoint Designer

1. Start Windows® Internet Explorer®.
2. On the **Site Actions** menu, click **Edit in SharePoint Designer**.

   The Microsoft SharePoint Designer starts.
3. In the Site Objects pane, click **External Content Types**.
4. On the ribbon, click **External Content Type**.
5. In the **External Content Type Information** section, next to **Name**, click **New external content type**.
6. Type **Training Suppliers**.

7. In the **External Content Type Operations** section, click **Click here to discover external data sources and define operations**.
8. Click **Add Connection**.
9. In the **Data Source Type** drop-down list, click **SQL Server**, and then click **OK**.
10. In the **Database Server** text box, type **SHAREPOINT\SharePoint**.
11. In the **Database Name** text box, type **HR Training Management**.
12. Click the **Connect with User's Identity** option, and then click **OK**.

▶ Task 3: Create an external content type

1. In the **Data Source Explorer** section, expand **HR Training Management**.
2. Expand **Tables**.
3. Right-click **Training Supplier**, and then click **Create All Operations**.

   The All Operations dialog box appears.
4. Click **Next**.
5. In the **Data Source Elements** section, click **Supplier ID**, but do not clear the check box.
6. Ensure that the **Map to Identifier** check box is selected.
7. Select the **Show In Picker** check box.
8. In the **Data Source Elements** section, click **Supplier**, but do not clear the check box.
9. Select the **Show In Picker** check box.
10. In the **Data Source Elements** section, click **Address**, but do not clear the check box.
11. Select the **Show In Picker** check box.
12. In the **Data Source Elements** section, click **Primary Contact**, but do not clear the check box.
13. Select the **Show In Picker** check box.
14. In the **Data Source Elements** section, click **Contact Telephone**, but do not clear the check box.
15. Select the **Show In Picker** check box.
16. In the **Data Source Elements** section, click **Contact Email**, but do not clear the check box.
17. Select the **Show In Picker** check box.
18. Click **Next**.
19. Click **Finish**.
20. On the **Quick Access** toolbar, click **Save**.

▶ Task 4: Create a list and form for the new external content type

1. On the ribbon, click **Create Lists & Form**.

   The Create List and Form for New External Content Type dialog box appears.
2. In the **List Name** text box, type **Training Suppliers**, and then click **OK**.

Task 5: Secure and test the external list

1. On the **Start** menu, click **All Programs**, click **Microsoft SharePoint 2010 Products**, and then click **SharePoint 2010 Central Administration**.
2. In the **Application Management** section, click **Manage service applications**.
3. Click **Business Data Connectivity Service**.
4. Point to the **TrainingSuppliers** link, and then click the drop-down arrow that appears.
5. Click **Set Permissions**.
6. Click the **Browse** icon.
7. Click **All Users**.
8. Click **All Authenticated Users**.
9. Click **Add**, and then click **OK**.
10. Click **Add**.
11. Select the **Edit** check box.
12. Select the **Execute** check box.
13. Select the **Selectable In Clients** check box.
14. Select the **Set Permissions** check box, and then click **OK**.
15. Navigate to **http://sharepoint**.

    The Training Suppliers link appears on the Quick Launch bar.
16. Click **Training Suppliers**.
17. Click **Graphic Design Institute**.
18. On the ribbon, click **Edit Item**.
19. In the **Address** text box, change **Chicago** to **New York**.
20. Click **Save**.
21. On the ribbon, click **New Item**.
22. In the **Supplier** text box, type **Proseware, Inc**.
23. In the **Address** text box, type **789 Main Street, Los Angeles**.
24. In the **PrimaryContact** text box, type **Christiane Jensen**.
25. In the **ContactTelephone** text box, type **444-4444**.
26. In the **ContactEmail** text box, type **ChristianeJensen@proseware.com**.
27. Click **Save**.
28. Close all applications.

Results: After this exercise, you should have created a connection, an external content type, a list, and the default forms for the **TrainingSupplier** table in the **HRTrainingManagement** external database. You should also have edited and added to the data from a SharePoint site.

Exercise 2: Creating Business Data Catalog Models by Using Visual Studio 2010

Scenario

In this exercise, you use Visual Studio 2010 to create a Business Data Catalog model to provide access to data stored in an external SQL Server database. The data operations that need to be performed include updating complex relationships.

The main tasks for this exercise are as follows:

1. Create a SharePoint project by using Visual Studio.
2. Add a Business Data Catalog model to the project and configure it.
3. Define type descriptors for the **ReadItem** method.
4. Define type descriptors for the **ReadList** method.
5. Create and configure data operation methods.
6. Implement the methods for the **Training eventEntity** entity.
7. Deploy and test the solution.

▶ Task 1: Create an empty SharePoint project

1. On the **Start** menu, click **All Programs**, open Microsoft Visual Studio 2010 by clicking **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.
2. On the **File** menu, point to **New**, and then click **Project**.
3. In the **Installed Templates** section, expand the **Visual C#** node, expand the **SharePoint** node, and then click **2010**.
4. Click **Empty SharePoint Project**.
5. In the **Name** text box, type **Lab06**.
6. In the **Location** text box, type **E:\Student\Lab06\Starter**, and then click **OK**.

   The SharePoint Customization Wizard appears.
7. In the **What local site do you want to use for debugging?** text box, type **http://sharepoint**.
8. In the **What is the trust level for this SharePoint solution?** section, click the **Deploy as a farm solution** option.
9. Click **Finish**.

   The project is created.

▶ Task 2: Add and configure a Business Data Connectivity model

1. In the Solution Explorer window, right-click **Lab06**, point to **Add**, and then click **New Item**.
2. Click **Business Data Connectivity Model**.
3. In the **Name** text box, type **Training event**, and then click **Add**.
4. In the Solution Explorer window, right-click **Entity1.cs**, and then click **Rename**.
5. Type **Training event.cs**, and then press ENTER.

6. In the **Microsoft Visual Studio** dialog box that prompts you to rename all references to Entity1, click **Yes**. If you are prompted to reload the file, click **Yes**.
7. Double-click **Training event.cs**.

   The Training event.cs code file appears.
8. Replace the existing property declarations in the class with the following:

```
public Int32 Training eventID { get; set; }
public DateTime EventDate { get; set; }
public string Status { get; set; }
public string LoginName { get; set; }
public string Title { get; set; }
public string EventType { get; set; }
public string Description { get; set; }
```

9. In the Solution Explorer window, double-click **Training event.bdcm**.
10. On the design surface, click **Entity1**.
11. On the **View** menu, click **Properties Window**.
12. In the Properties window, click **Name**.
13. Type **Training eventEntity**, and press ENTER.
14. On the design surface, click **Identifier1**.
15. In the Properties window, click **Name**.
16. Type **Training eventID**, and press ENTER.
17. In the Properties window, click **Type Name**.
18. In the drop-down list for the **Type Name** property, click **System.Int32**.
19. On the **File** menu, click **Save All**.

▶ Task 3: Define type descriptors for the ReadItem method

1. On the design surface, click **ReadItem**.
2. In the BDC Method Details pane, in the **Parameters** section below **ReadItem**, click **Identifier1** (in the **id** row). Then, click the drop-down arrow that appears and click **<Edit>**.
3. In the Properties window, click the **Name** property, type **Training eventID**, and then press ENTER.
4. In the Properties window, click **Type Name**.
5. In the drop-down list for the **Type Name** property, click **Int32**.
6. In the BDC Method Details pane, in the **Parameters** section below **ReadItem**, click **Entity1** (in the **return Parameter** row). Then, click the drop-down arrow that appears and click **<Edit>**.
7. In the Properties window, click the **Name** property, type **Training event**, and then press ENTER.
8. In the BDC Explorer window, expand the currently selected **Training event** node.
9. Beneath the **Training Event** node, click **Identifier1**.
10. In the Properties window, click the **Name** property, type **Training eventID**, and then press ENTER.
11. In the Properties window, click **Type Name**.
12. In the drop-down list for the **Type Name** property, click **Int32**.

13. In the Properties window, click **Read-only**.
14. In the drop-down list for the **Read-only** property, click **True**.
15. In the BDC Explorer window, click **Message**.
16. In the Properties window, click the **Name** property, type **EventDate**, and then press ENTER.
17. In the Properties window, click **Type Name**.
18. In the drop-down list for the **Type Name** property, click **Date Time**.
19. In the BDC Explorer window, right-click **Training Event**, and then click **Add Type Descriptor**.
20. In the Properties window, click the **Name** property, type **Status**, and then press ENTER.
21. In the BDC Explorer window, right-click **Training event**, and then click **Add Type Descriptor**.
22. In the Properties window, click the **Name** property, type **LoginName**, and then press ENTER.
23. In the BDC Explorer window, right-click **Training event**, and then click **Add Type Descriptor**.
24. In the Properties window, click the **Name** property, type **Title**, and then press ENTER.
25. In the BDC Explorer window, right-click **Training Event**, and then click **Add Type Descriptor**.
26. In the Properties window, click the **Name** property, type **EventType**, and then press ENTER.
27. In the BDC Explorer window, right-click **Training Event**, and then click **Add Type Descriptor**.
28. In the Properties window, click the **Name** property, type **Description**, and then press ENTER.
29. On the **File** menu, click **Save All**.

▶ Task 4: Define type descriptors for the Read List method

1. In the BDC Method Details pane, in the **Parameters** section below **ReadList**, click **Entity1List** (in the **return Parameter** row). Then, click the drop-down arrow that appears and click **<Edit>**.
2. In the Properties window, click the **Name** property, type **Training eventList**, and then press ENTER.
3. In the BDC Explorer window, expand the **Training event List** node.
4. Right-click **Entity1**, and then click **Delete**.
5. In the BDC Explorer window, below the **return Parameter** for the **Read Item** node, right-click **Training event**, and then click **Copy**.
6. In the BDC Explorer window, below the **return parameter** for the **Read List** node, right-click **Training even tList**, and then click **Paste**.
7. On the **File** menu, click **Save All**.

▶ Task 5: Create and configure data operation methods

1. In the BDC Method Details pane, collapse the **ReadList** and **ReadItem** nodes.
2. Click <**Add a Method>**, and then in the drop-down list, click **Create Creator Method**.
3. In the **Parameters** section below **Create**, click **NewTraining eventEntity** in the **Type Descriptor** column for the **newTraining eventEntity** row.
4. Click the drop-down arrow, and then click **<Edit>**.

5. In the BDC Explorer window, expand the selected **NewTraining eventEntity** node, and then click **Training eventID**.
6. In the Properties window, click **Read-only**, click the drop-down arrow, and then click **False**.
7. Collapse the **Create** node.
8. Click <**Add a Method>**, and then in the drop-down list, click **Create Deleter Method**.
9. Collapse the **Delete** node.
10. Click <**Add a Method>**, and then in the drop-down list, click **Create Updater Method**.
11. In the **Parameters** section below **Update**, click **Training eventEntity** in the **Type Descriptor** column for the **training eventEntity** row.
12. Click the drop-down arrow, and then click **<Edit>**.
13. In the BDC Explorer window, expand the selected **Training eventEntity** node, and then click **Training eventID**.
14. In the Properties window, click **Read-only**, click the drop-down arrow, and then click **False**.
15. On the **File** menu, click **Save All**.

▶ Task 6: Implement the methods for the Training eventEntity entity

1. On the design surface, double-click **ReadItem**.

   The code file for the **Training eventEntityService.cs** class appears.

2. Add the following using statements above the namespace declaration:

```
using System.Data;
using System.Data.Sql;
using System.Data.SqlClient;
using System.Data.SqlTypes;
```

3. Place the cursor *after* the *opening* brace of the class definition that reads **public class Training eventEntityService**, and press ENTER.
4. Add the following function to the space you have just created:

```
static SqlConnection getSqlConnection()
{
  SqlConnection sqlConn = new SqlConnection(
    "Integrated Security=SSPI;Persist Security Info=False;"
    + "Initial Catalog=HRTrainingManagement;"
    +  @"Data Source=SHAREPOINT\SharePoint");
  return (sqlConn);
}
```

5. Replace the contents of the **ReadItem** method with the following code:

```
SqlConnection thisConn = null;
Training event evt = null;
try
{
  evt = new Training event();
  thisConn = getSqlConnection();
  thisConn.Open();
  SqlCommand thisCmd = new SqlCommand();
  thisCmd.CommandText = "SELECT e.Training eventID, s.LoginName,"
  + " t.Title, t.EventType, t.Description, e.EventDate, e.Status"
  + " FROM Student s"
  + " INNER JOIN Training event e ON s.StudentID = e.StudentID"
```

```
  + " INNER JOIN TrainingObjects t ON e.TrainingID = t.TrainingID"
  + " WHERE e.Training eventID = " + id.ToString();
  thisCmd.Connection = thisConn;
  SqlDataReader thisReader =
    thisCmd.ExecuteReader(CommandBehavior.CloseConnection);
  if (thisReader.Read())
  {
    evt.Training eventID = id;
    evt.LoginName = thisReader[1].ToString();
    evt.Title = thisReader[2].ToString();
    evt.EventType = thisReader[3].ToString();
    evt.Description = thisReader[4].ToString();
    evt.EventDate = DateTime.Parse(thisReader[5].ToString());
    evt.Status = thisReader[6].ToString();
  }
  else
  {
    evt.Training eventID = -1;
    evt.LoginName = "Data Not Found";
    evt.Title = "Data Not Found";
    evt.EventType = "Data Not Found";
    evt.Description = "Data Not Found";
    evt.EventDate = DateTime.MinValue;
    evt.Status = "Data Not Found";
  }
  thisReader.Close();
  return (evt);
}
catch
{
  evt.Training eventID = -1;
  evt.LoginName = "Data Not Found";
  evt.Title = "Data Not Found";
  evt.EventType = "Data Not Found";
  evt.Description = "Data Not Found";
  evt.EventDate = DateTime.MinValue;
  evt.Status = "Data Not Found";
  return (evt);
}
finally
{
    thisConn.Dispose();
}
```

6. Replace the contents of the **ReadList** method with the following code:

```
SqlConnection thisConn = null;
List<Training event> allEvents;
try
{
  thisConn = getSqlConnection();
  allEvents = new List<Training event>();
  thisConn.Open();
  SqlCommand thisCommand = new SqlCommand();
  thisCommand.Connection = thisConn;
  thisCommand.CommandText = "SELECT e.Training eventID, LoginName,"
  + " t.Title, t.EventType, t.Description, e.EventDate, e.Status"
  + " FROM Student s"
  + " INNER JOIN Training event e ON s.StudentID = e.StudentID"
  + " INNER JOIN TrainingObjects t ON e.TrainingID = t.TrainingID";
  SqlDataReader thisReader =
    thisCommand.ExecuteReader(CommandBehavior.CloseConnection);
  while (thisReader.Read())
  {
    Training event evt = new Training event();
```

```
    evt.Training eventID = int.Parse(thisReader[0].ToString());
    evt.LoginName = thisReader[1].ToString();
    evt.Title = thisReader[2].ToString();
    evt.EventType = thisReader[3].ToString();
    evt.Description = thisReader[4].ToString();
    evt.EventDate = DateTime.Parse(thisReader[5].ToString());
    evt.Status = thisReader[6].ToString();
    allEvents.Add(evt);
  }
  thisReader.Close();
  Training event[] eventList = new Training event[allEvents.Count];
  for (int evtCounter = 0;
    evtCounter <= allEvents.Count - 1;
    evtCounter++)
  {
    eventList[evtCounter] = allEvents[evtCounter];
  }
  return (eventList);
}
catch (Exception ex)
{
  Training event[] errEventList = new Training event[1];
  Training event errEvt = new Training event();
  errEvt.Training eventID = -1;
  errEvt.LoginName = ex.Message;
  errEvt.Title = ex.Message;
  errEvt.EventType = ex.Message;
  errEvt.Description = ex.Message;
  errEvt.EventDate = DateTime.MinValue;
  errEvt.Status = ex.Message;
  errEventList[0] = errEvt;
  return (errEventList);
}
finally
{
    thisConn.Dispose();
}
```

7. Replace the contents of the **Create** method with the following code:

```
SqlConnection thisConn = null;
try
{
  thisConn = getSqlConnection();
  thisConn.Open();
  string studentName = newTraining eventEntity.LoginName;
  string trainingTitle = newTraining eventEntity.Title;
  string trainingType = newTraining eventEntity.EventType;
  string trainingDescription = newTraining eventEntity.Description;
  DateTime trainingDate = newTraining eventEntity.EventDate;
  string trainingStatus = newTraining eventEntity.Status;
  int studentID = 0;
  SqlCommand studentCmd = new SqlCommand();
  studentCmd.Connection = thisConn;
  studentCmd.CommandText = "SELECT StudentID"
    + " FROM Student"
    + " WHERE LoginName='" + studentName + "'";
  SqlDataReader studentReader =
    studentCmd.ExecuteReader(CommandBehavior.Default);
  if (studentReader.Read())
  {
    studentID = int.Parse(studentReader[0].ToString());
    studentReader.Close();
  }
  else
```

```
{
  studentReader.Close();
  SqlCommand addStudentCommand = new SqlCommand();
  addStudentCommand.Connection = thisConn;
  addStudentCommand.CommandText =
    "INSERT Student(LoginName) VALUES('" + studentName + "')";
  addStudentCommand.ExecuteNonQuery();
  SqlCommand getNewStudentCmd = new SqlCommand();
  getNewStudentCmd.Connection = thisConn;
  getNewStudentCmd.CommandText = "SELECT StudentID"
    + " FROM Student"
    + " WHERE LoginName = '" + studentName + "'";
  SqlDataReader getNewStudentReader =
    getNewStudentCmd.ExecuteReader(CommandBehavior.Default);
  getNewStudentReader.Read();
  studentID = int.Parse(getNewStudentReader[0].ToString());
  getNewStudentReader.Close();
}
int trainingID = 0;
SqlCommand trainingCmd = new SqlCommand();
trainingCmd.Connection = thisConn;
trainingCmd.CommandText = "SELECT TrainingID"
  + " FROM TrainingObjects"
  + " WHERE Title = '" + trainingTitle + "'"
  + " AND EventType = '" + trainingType + "'"
  + " AND Description = '" + trainingDescription + "'";
SqlDataReader trainingReader =
  trainingCmd.ExecuteReader(CommandBehavior.Default);
if (trainingReader.Read())
{
  trainingID = int.Parse(trainingReader[0].ToString());
  trainingReader.Close();
}
else
{
  trainingReader.Close();
  SqlCommand addTrainingCommand = new SqlCommand();
  addTrainingCommand.Connection = thisConn;
  addTrainingCommand.CommandText =
    "INSERT TrainingObjects(Title, EventType, Description)"
    + " VALUES('" + trainingTitle + "','"
    + trainingType + "','"
    + trainingDescription + "')";
  addTrainingCommand.ExecuteNonQuery();
  SqlCommand getNewTrainingCmd = new SqlCommand();
  getNewTrainingCmd.Connection = thisConn;
  getNewTrainingCmd.CommandText = "SELECT TrainingID"
    + " FROM TrainingObjects"
    + " WHERE Title = '" + trainingTitle + "'"
    + " AND EventType = '" + trainingType + "'"
    + " AND Description = '" + trainingDescription + "'";
  SqlDataReader getNewTrainingReader =
    getNewTrainingCmd.ExecuteReader(CommandBehavior.Default);
  getNewTrainingReader.Read();
  trainingID = int.Parse(getNewTrainingReader[0].ToString());
  getNewTrainingReader.Close();
}
SqlCommand insertEventCommand = new SqlCommand();
insertEventCommand.Connection = thisConn;
insertEventCommand.CommandText = "INSERT Training event"
  + "(StudentID, TrainingID, EventDate, Status) VALUES("
  + studentID
  + ", " + trainingID
  + ", '" + trainingDate.ToShortDateString() + "'"
```

```
    + ", '" + trainingStatus + "')";
  insertEventCommand.ExecuteNonQuery();
  return (newTraining eventEntity);
}
finally
{
  thisConn.Dispose();
}
```

8. Replace the contents of the **Delete** method with the following code:

```
SqlConnection thisConn = null;
try
{
  thisConn = getSqlConnection();
  thisConn.Open();
  SqlCommand thisCommand = new SqlCommand();
  thisCommand.Connection = thisConn;
  thisCommand.CommandText =
    "DELETE Training event WHERE Training eventID = "
    + training eventID.ToString();
  thisCommand.ExecuteNonQuery();
}
finally
{
  thisConn.Dispose();
}
```

9. Replace the contents of the **Update** method with the following code:

```
SqlConnection thisConn = null;
try
{
  thisConn = getSqlConnection();
  thisConn.Open();
  int training eventID = training eventEntity.Training eventID;
  string studentName = training eventEntity.LoginName;
  string trainingTitle = training eventEntity.Title;
  string trainingType = training eventEntity.EventType;
  string trainingDescription = training eventEntity.Description;
  DateTime trainingDate = training eventEntity.EventDate;
  string trainingStatus = training eventEntity.Status;
  int studentID = 0;
  SqlCommand studentCmd = new SqlCommand();
  studentCmd.Connection = thisConn;
  studentCmd.CommandText = "SELECT s.StudentID, s.LoginName"
   + " FROM Student s"
   + " INNER JOIN Training event e ON s.StudentID = e.StudentID"
   + " WHERE e.Training eventID = " + training eventID.ToString();
  SqlDataReader studentReader =
    studentCmd.ExecuteReader(CommandBehavior.Default);
  studentReader.Read();
  if (studentReader[1].ToString() == studentName)
  {
    studentID = int.Parse(studentReader[0].ToString());
    studentReader.Close();
  }
  else
  {
    studentReader.Close();
    SqlCommand changeStudentCmd = new SqlCommand();
    changeStudentCmd.Connection = thisConn;
    changeStudentCmd.CommandText = "SELECT StudentID"
      + " FROM Student"
      + " WHERE LoginName = '" + studentName + "'";
```

```
    SqlDataReader changeStudentReader =
    changeStudentCmd.ExecuteReader(CommandBehavior.Default);
    if (changeStudentReader.Read())
    {
      studentID = int.Parse(changeStudentReader[0].ToString());
      changeStudentReader.Close();
    }
    else
    {
      changeStudentReader.Close();
      SqlCommand addStudentCommand = new SqlCommand();
      addStudentCommand.Connection = thisConn;
      addStudentCommand.CommandText =
       "INSERT Student(LoginName) VALUES('" + studentName + "')";
      addStudentCommand.ExecuteNonQuery();
      SqlCommand getNewStudentCmd = new SqlCommand();
      getNewStudentCmd.Connection = thisConn;
      getNewStudentCmd.CommandText = "SELECT StudentID"
        + " FROM Student"
        + " WHERE LoginName = '" + studentName + "'";
      SqlDataReader getNewStudentReader =
        getNewStudentCmd.ExecuteReader(CommandBehavior.Default);
      getNewStudentReader.Read();
      studentID = int.Parse(getNewStudentReader[0].ToString());
      getNewStudentReader.Close();
    }
  }
  int trainingID = 0;
  SqlCommand trainingCmd = new SqlCommand();
  studentCmd.Connection = thisConn;
  studentCmd.CommandText =
    "SELECT t.TrainingID, t.Title, t.EventType, t.Description"
    + " FROM TrainingObjects t"
    + " INNER JOIN Training event e ON t.TrainingID = e.TrainingID"
    + " WHERE e.Training eventID = " + training eventID.ToString();
  SqlDataReader trainingReader =
  studentCmd.ExecuteReader(CommandBehavior.Default);
  trainingReader.Read();
  if ((trainingReader[1].ToString() == trainingTitle)
    && (trainingReader[2].ToString() == trainingType)
    && (trainingReader[3].ToString() == trainingDescription))
  {
    trainingID = int.Parse(trainingReader[0].ToString());
    trainingReader.Close();
  }
  else
  {
    trainingReader.Close();
    SqlCommand changeTrainingCmd = new SqlCommand();
    changeTrainingCmd.Connection = thisConn;
    changeTrainingCmd.CommandText = "SELECT TrainingID"
    + " FROM TrainingObjects"
    + " WHERE Title = '" + trainingTitle + "'"
    + " AND EventType = '" + trainingType + "'"
    + " AND Description = '" + trainingDescription + "'";
    SqlDataReader changeTrainingReader =
      changeTrainingCmd.ExecuteReader(CommandBehavior.Default);
    if (changeTrainingReader.Read())
    {
      trainingID = int.Parse(changeTrainingReader[0].ToString());
      changeTrainingReader.Close();
    }
    else
    {
```

```
      changeTrainingReader.Close();
      SqlCommand addTrainingCommand = new SqlCommand();
      addTrainingCommand.Connection = thisConn;
      addTrainingCommand.CommandText = "INSERT TrainingObjects("
        + "Title, EventType, Description) VALUES('"
        + trainingTitle + "','"
        + trainingType + "','"
        + trainingDescription + "')";
      addTrainingCommand.ExecuteNonQuery();
      SqlCommand getNewTrainingCmd = new SqlCommand();
      getNewTrainingCmd.Connection = thisConn;
      getNewTrainingCmd.CommandText = "SELECT TrainingID"
        + " FROM TrainingObjects"
        + " WHERE Title = '" + trainingTitle + "'"
        + " AND EventType = '" + trainingType + "'"
        + " AND Description = '" + trainingDescription + "'";
      SqlDataReader getNewTrainingReader =
        getNewTrainingCmd.ExecuteReader(CommandBehavior.Default);
      getNewTrainingReader.Read();
      trainingID = int.Parse(getNewTrainingReader[0].ToString());
      getNewTrainingReader.Close();
    }
  }
  SqlCommand updateEventCommand = new SqlCommand();
  updateEventCommand.Connection = thisConn;
  updateEventCommand.CommandText = "UPDATE Training event"
    + " SET StudentID=" + studentID
    + ", TrainingID=" + trainingID
    + ", EventDate='" + trainingDate.ToShortDateString() + "'"
    + ", Status='" + trainingStatus + "'"
    + " WHERE Training eventID=" + training eventID;
  updateEventCommand.ExecuteNonQuery();
}
finally
{
  thisConn.Dispose();
}
```

10. On the **File** menu, click **Save All**.

▶ Task 7: Deploy and test the solution

1. In the Solution Explorer window, right-click **Lab06**, and then click **Deploy**.
2. On the **Start** menu, click **All Programs**, click **Microsoft SharePoint 2010 Products**, and then click **SharePoint 2010 Central Administration**.
3. In the **Application Management** section, click **Manage service applications**.
4. Click **Business Data Connectivity Service**.
5. Point to the **Training eventEntity** link, and then click the drop-down arrow that appears.
6. Click **Set Permissions**.
7. Click the **Browse** icon.
8. Click **All Users**.
9. Click **All Authenticated Users**.
10. Click **Add**, and then click **OK**.
11. Click **Add**.

12. Select the **Edit** check box.
13. Select the **Execute** check box.
14. Select the **Selectable In Clients** check box.
15. Select the **Set Permissions** check box, and then click **OK**.
16. Navigate to **http://sharepoint**.
17. On the **Site Actions menu**, click **More Options**.

    A list of templates appears.
18. Click **External List**.
19. Click **Create**.
20. In the **Name** text box, type **Training Events**.
21. In the **Data source configuration** section, click **Select External Content Type**.

    The External Content Type Picker dialog box appears.
22. Click **Training event**, and then click **OK**.
23. Click **Create**.

    The empty list appears.
24. On the ribbon, click the **Items** tab.
25. Click **New Item**.
26. In the **Training eventID** text box, type **1**.
27. In the **EventDate** text box, type **9/9/2011**.
28. In the **LoginName** text box, type **SHAREPOINT\LauraG**.
29. In the **Status** text box, type **Not Started**.
30. In the **Title** text box, type **SharePoint 101**.
31. In the **Description** text box, type **End User Course for SharePoint 2010**.
32. In the **EventType** text box, type **Instructor-Led Training**.
33. Click **Save**.

    The new item is added to the list.
34. Click the item that has just been added to the list.
35. On the ribbon, click **Edit Item**.
36. In the **EventType** text box, type **E-Learning**.
37. Click **Save**.
38. Click the list item.
39. On the ribbon, click **Delete Item**, and then click **OK**.

    The item is deleted.
40. Close all applications. You have now completed this lab.

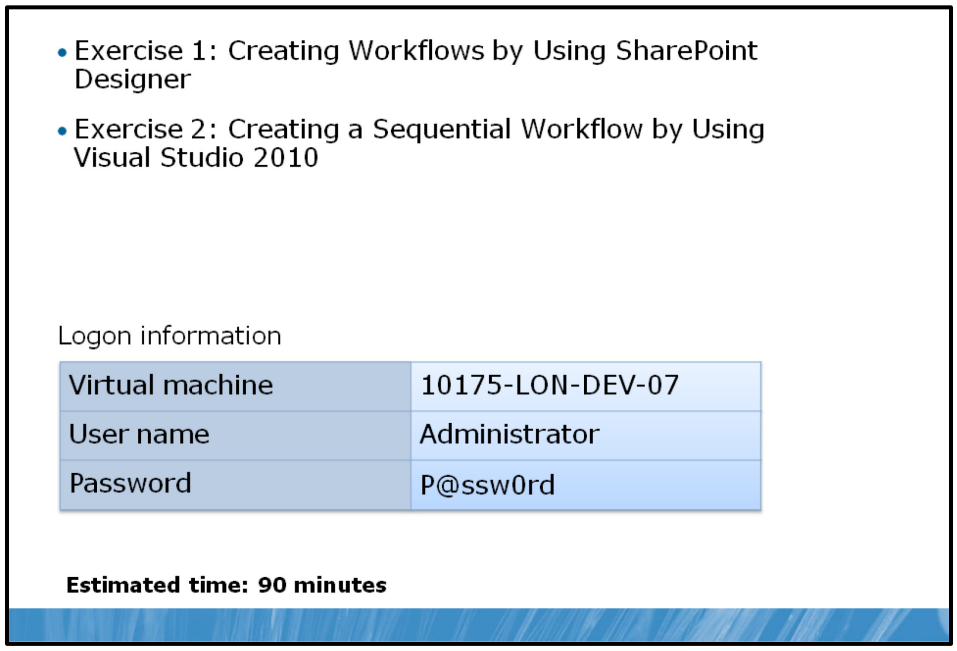
Results: After this exercise, you should have implemented a solution that provides access to complex data from the **HRTrainingManagement** database.

Module 7

Lab Instructions: Developing SharePoint 2010 Workflows

Contents:

Lab: Creating Workflows for SharePoint 2010



Log On to the Virtual Machine for This Lab

For this lab, you use the available virtual machine environment. Before you begin the lab, log on to the **10175-LON-DEV-07** virtual machine as **Administrator**, with the password **P@ssw0rd**.

Exercise 1: Creating Workflows by Using SharePoint Designer

Scenario

In this exercise, you use SharePoint Designer to create code-like logic for a workflow, without writing any code. The workflow includes parameters and variables, and makes decisions and performs actions based on the values of those parameters, variables, and values extracted from SharePoint list items.

The main tasks for this exercise are as follows:

1. Open the site in SharePoint Designer.
2. Create a new workflow with initiation parameters and variables.
3. Add actions and decision-making to the workflow.
4. Test the workflow.

▶ Task 1: Open the site in SharePoint Designer

1. Use Windows® Internet Explorer® to browse to **http://sharepoint**.
2. On the **Site Actions** menu, click **Edit in SharePoint Designer**.
3. In the **Site Objects** pane, click **Lists and Libraries**.
4. Click **Employment Contracts**.
5. In the **Settings** section, click **Require content approval for submitted items**.
6. On the **Quick Access Toolbar**, click **Save**.

▶ Task 2: Create a new workflow with initiation parameters and variables

1. In the **Workflows** section, click **New**.
2. In the **Name** text box, type **Approve Contract**.
3. Click **OK**.
4. On the ribbon, click **Initiation Form Parameters**.
5. Click **Add**.
6. In the **Field name** text box, type **ApproverEmail**.
7. In the **Information type** drop-down list, click **Person or Group**.
8. Click **Next**.
9. Click **Finish**.
10. Click **Add**.
11. In the **Field name** text box, type **MaxSalary**.
12. In the **Information type** drop-down list, click **Number (1, 1.0, 100)**.
13. Click **Next**.
14. In the **Default value** text box, type **120000**.
15. Click **Finish**.
16. Click **OK**.

17. On the ribbon, click **Local Variables**.
18. Click **Add**.
19. In the **Name** text box, type **Salary**.
20. In the **Type** drop-down list, click **Number**.
21. Click **OK**.
22. Click **OK**.

▶ Task 3: Add actions and decision-making to the workflow

1. On the ribbon, click **Action**, and then click **Set Workflow Variable**.
2. Click the **Workflow variable** link, and then click **Variable: Salary**.
3. Click the **value** link, and then click the **Define workflow lookup** button.
4. In the **Data source** drop-down list, click **Current Item**.
5. In the **Field from source** drop-down list, click **Salary**.
6. Click **OK**.
7. On the ribbon, click **Condition**, and then click **If any value equals value**.
8. Click the first **value** link, and then click the **Define workflow lookup** button.
9. In the **Data source** drop-down list, click **Workflow variables and parameters**.
10. In the **Field from source** drop-down list, click **Variable: Salary**.
11. Click **OK**.
12. Click the **equals** link, and then click **is greater than**.
13. Click the second **value** link, and then click the **Define workflow lookup** button.
14. In the **Data source** drop-down list, click **Workflow variables and parameters**.
15. In the **Field from source** drop-down list, click **Parameter: MaxSalary**.
16. Click **OK**.
17. Click **(Start typing or use the Insert group in the Ribbon)**.
18. On the ribbon, click **Action**, and then click **Set Field in Current Item**.
19. Click the **field** link, and then click **Review Comments**.
20. Click the **value** link, and then type **Auto-rejected: Salary breaches the current salary cap**. Then press ENTER.
21. On the ribbon, click **Action**, and then click **Set Content Approval Status**.
22. Click the **this status** link, and then click **Rejected**.
23. Click the **comments** link, and then type **Rejected by workflow**. Then press ENTER.
24. On the ribbon, click **Else-If Branch**.
25. On the ribbon, click **Action**, and then click **Set Content Approval Status**.
26. Click the **this status** link, and then click **Pending**.

27. Click the **comments** link, and then type **Awaiting review**. Then press ENTER.
28. On the ribbon, click **Action**, and then click **Set Field in Current Item**.
29. Click the **field** link, and then click **Reviewer**.
30. Click the **value** link.
31. Click **Workflow Lookup for a User**, and then click **Add**.
32. In the **Data source** drop-down list, click **Workflow variables and parameters**.
33. In the **Field from source** drop-down list, click **Parameter: ApproverEmail**.
34. In the **Return field as** drop-down list, click **Email Address**, then click **OK** and then click **OK** again.
35. On the **Quick Access Toolbar**, click **Save**.
36. On the ribbon, click **Publish**.
37. Close SharePoint Designer.

▶ Task 4: Test the workflow

1. Use Internet Explorer to browse to **http://sharepoint**.
2. In the **Quick Launch** bar, click **Employment Contracts**.
3. On the ribbon, click the **Documents** tab.
4. On the ribbon, click **Upload Document**.
5. Click **Browse**, and browse to the **E:\Student\Lab07\Documents** folder.
6. Click **Employment Contract Bobby Moore.docx**, and then click **Open**.
7. Click **OK**.
8. In the **Title** text box, type **Bobby Moore**.
9. In the **Salary** text box, type **130000**.
10. Click **Save**.
11. In the list, point to **Employment Contract Bobby Moore**, click the drop-down arrow, and then click **Workflows**.
12. Click **Approve Contract**.
13. In the **ApproverEmail** text box, type **krishnas@sharepoint.com**.
14. Click **Check Names**.
15. Click **Start**.

    The workflow runs, and sets the Review comments and Approval Status columns appropriately for a rejected contract.

16. In the list, point to **Employment Contract Bobby Moore**, click the drop-down arrow, and then click **Edit Properties**.
17. Edit the **Salary** text box so that it contains **125000**.
18. Delete the contents of the **Review Comments** text box.
19. Click **Save**.

20. In the list, point to **Employment Contract Bobby Moore**, click the drop-down arrow, and then click **Workflows**.
21. Click **Approve Contract**.
22. In the **ApproverEmail** text box, type **krishnas@sharepoint.com**.
23. Click **Check Names**.
24. Edit the **MaxSalary** text box so that it contains **125,500**.
25. Click **Start**.

    The workflow runs, and sets the Approval Status to pending and the Reviewer to krishnas@sharepoint.com.

26. Close all applications.

Results: After this exercise, you should have created and tested a workflow by using SharePoint Designer.

Exercise 2: Creating a Sequential Workflow by Using Visual Studio 2010

Scenario

In this exercise, you develop a sequential workflow and add code for making decisions based on the values in SharePoint list items.

The main tasks for this exercise are as follows:

1. Create an empty SharePoint project.
2. Add a sequential workflow.
3. Develop the sequential workflow.
4. Test the sequential workflow.

▶ Task 1: Create an empty SharePoint project

1. On the **Start** menu, click **All Programs**, click **Microsoft Visual Studio 2010** and then click **Microsoft Visual Studio 2010**.
2. On the **File** menu, point to **New**, and then click **Project**.
3. In the **Installed Templates** section, expand the **Visual C#** node, expand the **SharePoint** node, and then click **2010**.
4. Click **Empty SharePoint Project**.
5. In the **Name** text box, type **Lab07**.
6. In the **Location** text box, type **E:\Student\Lab07\Starter**.
7. Click **OK**.

   The SharePoint Customization Wizard appears.
8. In the **What local site do you want to use for debugging?** text box, type **http://sharepoint**.
9. In the **What is the trust level for this SharePoint** solution section, click the **Deploy as a farm solution** option.
10. Click **Finish**.

    The project is created.

▶ Task 2: Add a sequential workflow

1. In the Solution Explorer window, right-click **Lab07**, point to **Add** and then click **New Item**.
2. Click **Sequential Workflow**.
3. In the **Name** text box, type **reviewContracts** and then click **Add**.

   The SharePoint Customization Wizard appears.
4. In the **What is the name of the workflow?** text box, type **reviewContracts**.
5. Click **List Workflow**.
6. Click **Next**.
7. In the **The library or list to associate your workflow with** drop-down list, click **Employment Contracts**.

8. Click **Next**.
9. Be sure the **A user manually starts the workflow** checkbox is checked.
10. Be sure the **The workflow starts automatically when an item is created** checkbox is checked.
11. Be sure the **The workflow starts automatically when an item is changed** checkbox is *not* checked.
12. Click **Finish**.
13. On the **View** menu, click **Toolbox**.
14. Click the **Auto Hide** pin to pin the toolbox open.
15. In the toolbox, expand the **Windows Workflow v3.0** group.
16. Drag a **While** item from the toolbox, and drop it beneath the **onWorkflowActivated1** item on the workflow design surface.
17. In the toolbox, expand the **SharePoint Workflow** group.
18. Drag an **OnWorkflowItemChanged** item from the toolbox, and drop it on the **Drop an Activity Here** text in the **whileActivity1** item.
19. Click the **onWorkflowActivated1** item on the workflow design surface.
20. In the Properties window, click **Invoked**.
21. Type **onWorkflowActivated** and then press ENTER.

    Visual Studio displays the code file.
22. Click the **reviewContracts.cs [Design]** tab.
23. Click the **whileActivity1** item on the workflow design surface. (Be sure you click the **whileActivity1** item, and not the **onWorkflowItemChanged1** item.)
24. In the Properties window, click **Conditions**, and then click the drop-down arrow next to **(None)**. Then click **Code Condition**.
25. Expand the **Condition** property, and then click the **Condition** row that appears.
26. Type **isWorkflowPending**, and then press ENTER.

    Visual Studio displays the code file.
27. Click the **reviewContracts.cs [Design]** tab.
28. Click the **onWorkflowItemChanged1** item on the workflow design surface. (Be sure you click the **onWorkflowItemChanged1** item, and not the **whileActivity1** item.)
29. In the Properties window, click **CorrelationToken**, click the drop-down arrow, and then click **workflowToken**.
30. In the Properties window, click **Invoked**.
31. Type **onWorkflowItemChanged** and press ENTER.
32. On the **File** menu, click **Save All**.

▶ Task 3: Develop the sequential workflow

1. On the line above the **onWorkflowActivated** method, add the following code:

```
bool wfPending = true;
```

2. Create an empty line after the code you have just typed.
3. Add the following code to the space you have just created:

```
void checkDocStatus()
{
   if(workflowProperties.Item["Contract Status"].ToString()
      == "Review Complete")
  {
    wfPending = false;
  }
}
```

4. Add the following code between the opening and closing braces of the **onWorkflowActivated** method:

```
checkDocStatus();
```

5. Add the following code between the opening and closing braces of the **isWorkflowPending** method:

```
e.Result = wfPending;
```

6. Add the following code between the opening and closing braces of the **onWorkflowItemChanged** method:

```
checkDocStatus();
```

▶ Task 4: Test the sequential workflow

1. In the Solution Explorer window, right-click **Lab07** and then click **Deploy**.
2. Use Internet Explorer to browse to **http://sharepoint**.
3. In the **Quick Launch** bar, click **Employment Contracts**.
4. On the ribbon, click the **Documents** tab.
5. Click **Upload Document**.
6. Click **Browse**, and browse to the **E:\Student\Lab07\Documents** folder.
7. Click **Employment Contract Paula Bento.docx**, and then click **Open**.
8. Click **OK**.
9. In the **Title** text box, type **Paula Bento**.
10. In the **Contract Status** drop-down list, click **Review Needed**.
11. In the **Salary** text box, type **120000**.
12. Click **Save**.
13. Scroll to the right of the list, and note that the **ReviewContracts** workflow is **In Progress**.
14. In the list, point to **Employment Contract Paula Bento**, click the drop-down arrow, and then click **Edit Properties**.
15. In the **Contract Status** drop-down list, click **Review Complete**.
16. Click **Save**.
17. In the **Quick Launch** bar, click **Employment Contracts** to refresh the view of the list.
18. Scroll to the right of the list, and note that the **ReviewContracts** workflow is **Completed**.

19. Close all applications.

    You have now completed this lab.

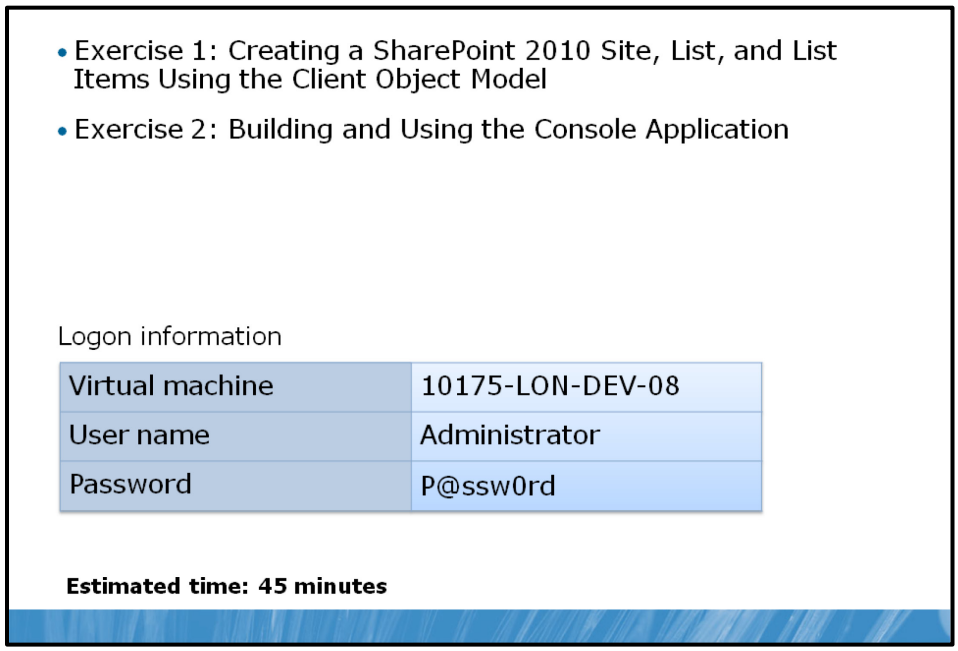
Results: After this exercise, you should have created and tested a workflow by using Visual Studio 2010.

Module 8

Lab Instructions: Working with Client-Based APIs for SharePoint 2010

Contents:

Lab: Developing .NET Applications by Using the SharePoint Client Object Model



Log On to the Virtual Machine for This Lab

For this lab, you use the available virtual machine environment. Before you begin the lab, you must log on to the **10175-LON-DEV-08** virtual machine as **Administrator** using the password **P@ssw0rd**.

Exercise 1: Creating a SharePoint 2010 Site, List, and List Items Using the Client Object Model

Scenario

The HR department has decided to implement a skills management application that associates skills and required levels with jobs in the organization.

In this exercise, you create the Skills Matrix site for your SharePoint site collection. You create the site from a console application using the SharePoint 2010 Client Object Model.

Your console application prompts users for their chosen action. If the action is Create, the application then prompts for the Url of the site in which the Skills Matrix subsite will be created.

The application creates a site named Skills Matrix and then uses the SharePoint Client Object Model to create lists titled Jobs and Skills that are then added to the Quick Launch menu. In addition, the application creates a list called Mashup that is hidden from users.

Following the creation of the lists, the application creates list items and populates the Jobs list and the Skills list. Later labs use the Mashup list to store the mappings of skills to jobs at the required level.

The main tasks for this exercise are as follows:

1. Create a console application.
2. Add a reference to the Microsoft SharePoint Client Object Model DLLs.
3. Add a using reference for the SharePoint Client Object Model.
4. Add code to the Main() method.
5. Create the createSkillsMatrix() Method.

▶ Task 1: Create a console application

1. On the Start menu, click All Programs, click Microsoft Visual Studio 2010, and then click Microsoft Visual Studio 2010.
   Microsoft Visual Studio 2010 opens.
2. On the **File** menu, point to **New**, and then click **Project**.
3. In the **Installed Templates** section, click the **Visual C#** node, and then click **Console Application**.
4. In the **.NET Framework version** drop-down list, click .Net Framework 3.5.
5. In the **Name** text box, type **Lab08**.
6. In the **Location** text box, type **E:\Student\Lab08\Starter**, and then click **OK**.

   The project is created.

▶ Task 2: Add a reference to the Microsoft SharePoint Client Object Model DLLs

1. In the Solutions Explorer window, right-click the **References** node, and click **Add Reference**.
2. In the Add Reference dialog box, click the Browse tab, and then browse to the following location:
   C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\14\ISAPI
3. Click Microsoft.SharePoint.Client.dll, hold down the CTRL key, click Microsoft.SharePoint.Client.Runtime.dll, and then click OK.

   The references to the Microsoft SharePoint Client Object Model DLLs are added to the Solution Explorer window under **References**.

▶ Task 3: Add a using reference for the SharePoint Client Object Model

- At the top of the code file, after the last using statement, add the following code:

```
using Microsoft.SharePoint.Client;
```

▶ Task 4: Add code to the Main() method

- In the code window, add the following code between the braces for the **Main** method:

```
string option = string.Empty;
Console.Write("Options: [C] Create Skills Matrix: | [V] View Skills Matrix: ");
option = Console.ReadLine().ToUpper();
while ((option != "C") && (option != "V"))
{
  Console.Write(
    "Options: [C] Create Matrix: | [V] View Matrix: ");
  option = Console.ReadLine().ToUpper();
}
if (option == "C")
{
  createSkillsMatrix();
  return;
}
```

▶ Task 5: Create the createSkillsMatrix() Method

- In the code window, after the closing brace for the **Main()** method, add the following code:

```
static void createSkillsMatrix()
{
  ClientContext remoteCtx = null;
  try
  {
    Console.Write(
      "Enter the URL of a SharePoint site to house the Skills Matrix"
      + " (e.g. http://myserver): ");
    string siteUrl = Console.ReadLine().ToLower();
    while (!siteUrl.StartsWith("http://"))
    {
      Console.Write(
       "Enter the URL of a SharePoint site to house the Skills Matrix"
       + " (e.g. http://myserver): ");
      siteUrl = Console.ReadLine().ToLower();
    }
    remoteCtx = new ClientContext(siteUrl);
    Web remoteWeb = remoteCtx.Web;
    remoteCtx.Load(remoteWeb);
    WebCreationInformation skillsInfo = new WebCreationInformation();
    skillsInfo.Title = "Skills Matrix";
    skillsInfo.Language = 1033;
    skillsInfo.WebTemplate = "STS#1";
    skillsInfo.Url = "Skills";
    skillsInfo.UseSamePermissionsAsParentSite = true;
    Web skillsWeb = remoteWeb.Webs.Add(skillsInfo);
    skillsWeb.QuickLaunchEnabled = true;
    Console.Write("Skills Site is being created...");
    remoteCtx.ExecuteQuery();
    Console.WriteLine(" Success!");
    Console.Write("Skills Matrix is being created...");
    ListCreationInformation jobDefInfo =
      new ListCreationInformation();
    jobDefInfo.TemplateType = (int)ListTemplateType.GenericList;
    jobDefInfo.Title = "Jobs";
    jobDefInfo.QuickLaunchOption = QuickLaunchOptions.On;
```

```
List jobDef = skillsWeb.Lists.Add(jobDefInfo);
jobDef.Fields.AddFieldAsXml(
  @"<Field Type='Text' DisplayName='Description'/>",true,
  AddFieldOptions.DefaultValue);
jobDef.Fields.AddFieldAsXml(
  @"<Field Type='Choice' DisplayName='Level' Format='Dropdown'>"
  + "<Default>Junior</Default>"
  + "<CHOICES>"
  + "<CHOICE>N/A</CHOICE>"
  + "<CHOICE>Junior</CHOICE>"
  + "<CHOICE>Senior</CHOICE>"
  + "<CHOICE>Lead</CHOICE>"
  + "</CHOICES>"
  + "</Field>", true, AddFieldOptions.DefaultValue);
ListItemCreationInformation jobInfo =
  new ListItemCreationInformation();
ListItem job = jobDef.AddItem(jobInfo);
job["Title"] = "Developer";
job["Description"] = "Writes Code to Spec";
job["Level"] = "Junior";
job.Update();
job = jobDef.AddItem(jobInfo);
job["Title"] = "Developer";
job["Description"] = "Creates Specs and Writes Code";
job["Level"] = "Senior";
job.Update();
job = jobDef.AddItem(jobInfo);
job["Title"] = "Developer";
job["Description"] = "Manages Development Teams";
job["Level"] = "Lead";
job.Update();
job = jobDef.AddItem(jobInfo);
job["Title"] = "Analyst";
job["Description"] = "Defines business processes";
job["Level"] = "Junior";
job.Update();
job = jobDef.AddItem(jobInfo);
job["Title"] = "Analyst";
job["Description"] =
  "Defines business processes and creates reports";
job["Level"] = "Senior";
job.Update();
job = jobDef.AddItem(jobInfo);
job["Title"] = "Analyst";
job["Description"] = "Manages Analysis Teams";
job["Level"] = "Lead";
job.Update();
ListCreationInformation skillDefInfo =
  new ListCreationInformation();
skillDefInfo.TemplateType = (int)ListTemplateType.GenericList;
skillDefInfo.Title = "Skills";
skillDefInfo.QuickLaunchOption = QuickLaunchOptions.On;
List skillDef = skillsWeb.Lists.Add(skillDefInfo);
skillDef.Fields.AddFieldAsXml(
  @"<Field Type='Text' DisplayName='Description'/>",
  true, AddFieldOptions.DefaultValue);
skillDef.Fields.AddFieldAsXml(
  @"<Field Type='Choice' DisplayName='Importance'"
  + " Format='Dropdown'>"
  + "<Default>Essential</Default>"
  + "<CHOICES>"
  + "<CHOICE>Essential</CHOICE>"
  + "<CHOICE>Preferred</CHOICE>"
  + "<CHOICE>Optional</CHOICE>"
```

```
    + "</CHOICES>"
    + "</Field>", true, AddFieldOptions.DefaultValue);
ListItemCreationInformation skillInfo =
  new ListItemCreationInformation();
ListItem skill = skillDef.AddItem(skillInfo);
skill["Title"] = "Write Code";
skill["Importance"] = "Essential";
skill.Update();
skillInfo = new ListItemCreationInformation();
skill = skillDef.AddItem(skillInfo);
skill["Title"] = "Write Code";
skill["Importance"] = "Preferred";
skill.Update();
skillInfo = new ListItemCreationInformation();
skill = skillDef.AddItem(skillInfo);
skill["Title"] = "Write Code";
skill["Importance"] = "Optional";
skill.Update();
skill = skillDef.AddItem(skillInfo);
skill["Title"] = "Make Technical Decisions";
skill["Importance"] = "Essential";
skill.Update();
skillInfo = new ListItemCreationInformation();
skill = skillDef.AddItem(skillInfo);
skill["Title"] = "Make Technical Decisions";
skill["Importance"] = "Preferred";
skill.Update();
skillInfo = new ListItemCreationInformation();
skill = skillDef.AddItem(skillInfo);
skill["Title"] = "Make Technical Decisions";
skill["Importance"] = "Optional";
skill.Update();
skill = skillDef.AddItem(skillInfo);
skill["Title"] = "Analyze Business Processes";
skill["Importance"] = "Essential";
skill.Update();
skillInfo = new ListItemCreationInformation();
skill = skillDef.AddItem(skillInfo);
skill["Title"] = "Analyze Business Processes";
skill["Importance"] = "Preferred";
skill.Update();
skillInfo = new ListItemCreationInformation();
skill = skillDef.AddItem(skillInfo);
skill["Title"] = "Analyze Business Processes";
skill["Importance"] = "Optional";
skill.Update();
skill = skillDef.AddItem(skillInfo);
skill["Title"] = "Write Reports";
skill["Importance"] = "Essential";
skill.Update();
skillInfo = new ListItemCreationInformation();
skill = skillDef.AddItem(skillInfo);
skill["Title"] = "Write Reports";
skill["Importance"] = "Preferred";
skill.Update();
skillInfo = new ListItemCreationInformation();
skill = skillDef.AddItem(skillInfo);
skill["Title"] = "Write Reports";
skill["Importance"] = "Optional";
skill.Update();
remoteCtx.ExecuteQuery();
remoteCtx.Load(jobDef);
remoteCtx.Load(skillDef);
remoteCtx.ExecuteQuery();
```

```
    ListCreationInformation mixerInfo = new ListCreationInformation();
    mixerInfo.TemplateType = (int)ListTemplateType.GenericList;
    mixerInfo.Title = "Mashup";
    mixerInfo.QuickLaunchOption = QuickLaunchOptions.Off;
    List mixer = skillsWeb.Lists.Add(mixerInfo);
    mixer.Fields.AddFieldAsXml(@"<Field Type='Integer' "
    + " Name='LookupJobs' DisplayName='LookupJobs' Required='TRUE'/>",
      true, AddFieldOptions.DefaultValue);
    mixer.Fields.AddFieldAsXml(
      @"<Field Type='Integer' Name='LookupSkills' "
      + "DisplayName='LookupSkills' Required='TRUE'/>",
      true, AddFieldOptions.DefaultValue);
    Field titleField = mixer.Fields.GetByInternalNameOrTitle("Title");
    titleField.Hidden = true;
    titleField.Update();
    mixer.Hidden = true;
    mixer.Update();
    remoteCtx.ExecuteQuery();
    Console.WriteLine(" Success!");
    Console.Write("Press any key to exit...");
    Console.ReadLine();
  }
  catch(Exception ex)
  {
    Console.WriteLine();
    Console.WriteLine(ex.Message);
    Console.Write("Press any key to exit...");
    Console.ReadLine();
  }
  finally
  {
    remoteCtx.Dispose();
  }
}
```

Results: After this exercise, you have completed the code for creating a skills subsite.

Exercise 2: Building and Using the Console Application

▶ Task 1: Build the console application

1. In the Solution Explorer window, right-click the **Lab08** project, and click **Build**.

   Verify the Build Succeeded message appears below the code pane.

2. On the **Debug menu**, click Start Without Debugging.
3. In the console window, review the text and press C.
4. At the Enter the URL of a SharePoint site to house the Skills Matrix prompt, type http://SharePoint, and then press ENTER.
5. When you are notified that the Skills Site and the Skills Matrix have been created, press ENTER twice to close the console application.

▶ Task 2: Review the Skills Matrix site

1. Open Windows® Internet Explorer® and browse to **http://sharepoint**.
2. On the Quick Launch menu, click All Site Content.
3. In the Sites and Workspaces section, click Skills Matrix.
4. On the **Quick Launch** menu, click **Skills**.

   Review the skills that were created in the console application.

5. On the **Quick Launch** menu, click **Jobs**.

   Review the Jobs that were created in the console application.

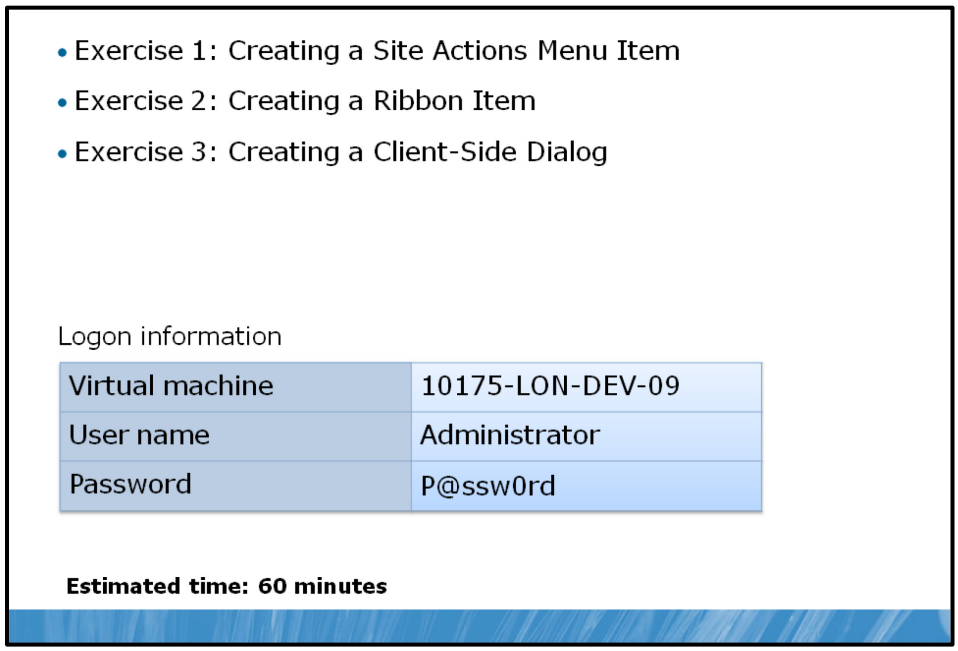
Results: After this exercise, you should have a subsite in SharePoint called Skills Matrix.
A list called Jobs and a list called Skills are available from the Quick Launch menu of the subsite.
A list called Mashup has also been created, but it is hidden.

Module 9

Lab Instructions: Developing Interactive User Interfaces

Contents:

Lab: Developing User Interface Components for SharePoint 2010 Solutions



Log On to the Virtual Machine for This Lab

For this lab, you use the available virtual machine environment. Before you begin the lab, log on to the **10175-LON-DEV-09** virtual machine as **Administrator**, with a password **P@ssw0rd**.

Exercise 1: Creating a Site Actions Menu Item

Scenario

In this exercise, you add a new feature to the SharePoint project that adds a new menu option to the Site Actions menu. The new menu option provides a direct link to the manage features option in SharePoint Site Settings.

▶ Task 1: Create an empty SharePoint Project

1. On the **Start** menu, click **All Programs**, click **Microsoft Visual Studio 2010** and then click **Microsoft Visual Studio 2010**.
2. On the **File** menu, point to **New**, and then click **Project**.
3. In the **Installed Templates** section, expand the **Visual C#** node, expand the **SharePoint** node and then click **2010**.
4. Click **Empty SharePoint Project**.
5. In the **Name** text box, type **Lab09**.
6. In the **Location** text box, type **E:\Student\Lab09\Starter**.
7. Click **OK**.

   The SharePoint Customization Wizard appears.
8. In the **What local site do you want to use for debugging?** text box, type **http://sharepoint**.
9. In the **What is the trust level for this SharePoint** solution section, click the **Deploy as a farm solution** option.
10. Click **Finish**.

    The project is created.

▶ Task 2: Add an empty element to the project

1. In the Solution Explorer window, right-click **Lab09** and click **Add**, and then click **New Item**.
2. In the **Add New Item** dialog, in the list of installed templates, click **Empty Element**.
3. In the **Name** textbox, type **SiteActions_Features** and click **Add**.

   The SiteActions_Features node is added to the project and the Elements.xml file opens in the code window.

▶ Task 3: Edit the Elements.xml file to add a new menu option to the Site Actions menu

- Replace the elements.xml file with the following markup:

```
<?xml version="1.0" encoding="utf-8"?>
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
  <CustomAction Id="NewUIActionsMenu"
    GroupId="SiteActions"
      Location="Microsoft.SharePoint.StandardMenu"
      Sequence="1970"
      Title="Manage Features">
    <UrlAction Url="/_layouts/ManageFeatures.aspx" />
  </CustomAction>
</Elements>
```

Task 4: Build and test the item

1. In the Solution Explorer window, right-click the **Lab09** project and click **Deploy**.
2. Start **Internet Explorer** and browse to the site **http://sharepoint**.
3. On the **Site Actions** menu, click **Manage Features**.

   The new option called Manage Features appears on the Site Actions menu and navigates the user to /_layouts/ManageFeatures.aspx.
4. On the **ManageFeatures.aspx** page, locate the **Lab09 Feature1** item and click **Deactivate**.
5. On the **You are about to deactivate the Lab09 Feature1 feature** warning message, click **Deactivate this feature**.
6. Click the **Site Actions** menu and review the menu items.

   Note that the Manage Features option no longer appears on this list.
7. On the **ManageFeatures.aspx** page, locate the **Lab09 Feature1** item and click **Activate**.

   The Manage Features option is re-enabled on the menu.

Results: After this exercise, you should have created a new feature that adds or removes an option from the Site Actions menu. You should have provided a Url for that menu option that directs the user directly to the Manage Features page in the SharePoint Site Settings area.

Exercise 2: Creating a Ribbon Item

Scenario

In this exercise, you create a new ribbon toolbar option called Help Videos to appear on the document ribbon. You specify the ribbon item in XML and create a SharePoint Images folder and upload the icon image to your Visual Studio project before deployment.

The main tasks for this exercise are as follows:

1. Add an empty element to the project.
2. Specify the new document ribbon item in XML.
3. Add an images folder to the Visual Studio project and then upload an existing image to that folder.
4. Deploy the project to SharePoint 2010 and test the new ribbon.

▶ Task 1: Add an empty element to the project

1. Switch back to the Visual Studio 2010 Lab09 project.
2. In the Solution Explorer window, right-click **Lab09**, click **Add** and then click **New Item**.
3. Click **Empty Element**.
4. In the **Name** textbox, type **HelpRibbon** and click **Add**.

   The HelpRibbon node is added to the project and the Elements.xml file opens in the code window.

▶ Task 2: Edit the Elements.xml file to add a new item to the Documents ribbon

1. Replace the Elements.xml file with the following markup:

```
<?xml version="1.0" encoding="utf-8"?>
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
  <CustomAction
  Id="NewUIRibbonControl"
  RegistrationType="List"
  RegistrationId="101"
  Location="CommandUI.Ribbon">
  <CommandUIExtension>
    <CommandUIDefinitions>
      <CommandUIDefinition
        Location="Ribbon.Documents.New.Controls._children">
        <Button
          Id="NewUIRibbonControl.ShowHelp"
          Alt="Help"
          Sequence="1981"
          Command="ShowHelp"
          Image32by32="/_layouts/images/lab09/dochelp.png"
          LabelText="Help Videos"
          TemplateAlias="o1"/>
        </CommandUIDefinition>
      </CommandUIDefinitions>
      <CommandUIHandlers>
        <CommandUIHandler
          Command="ShowHelp"
          CommandAction="javascript:window.open(
'http://msdn.microsoft.com/en-gb/sharepoint/ee663870.aspx');" />
      </CommandUIHandlers>
    </CommandUIExtension>
  </CustomAction>
</Elements>
```

2. On the **File** menu, click **Save All**.

Task 3: Add images to the SharePoint project

1. In the Solution Explorer window, right-click **Lab09**, click **Add** and then click **SharePoint Images Mapped Folder**.
2. In the Solution Explorer window, under the **Images** node, right-click **Lab09** and click **Add** and then click **Existing Item**.
3. Locate the file **E:\Student\Lab09\docHelp.png** and click **Add**.

Task 4: Build and test the ribbon item

1. In the Solution Explorer window, right-click the **Lab09** project and click **Deploy**.
2. Switch back to **Internet Explorer** and browse to the site **http://sharepoint**.
3. On the **Quick Launch** bar, click **Shared Documents**.
4. On the ribbon, click the **Documents** tab.
5. On the ribbon, in the **New** section, click **Help Videos**.

   This is the ribbon item that you have just added.
6. Close all instances of Internet Explorer.

Results: After this exercise, you should have created a new ribbon item that appears on the Documents library in the New section.

Exercise 3: Creating a Client-Side Dialog

Scenario

In this exercise, you create a new Visual Web Part that displays items from the Jobs list in the Skills Matrix sub-site in a Treeview control on the Web Part.

You add sub-nodes to the Treeview for skills that have been mapped to the job in the mashup list that is also in the Skills Matrix sub-site.

The main tasks for this exercise are as follows:

1. Add a Visual Web Part to the project.
2. Add a Treeview control to the design surface.
3. Construct code to return Jobs and mapped skills from the Skills Matrix sub-site.

▶ Task 1: Add a Visual Web Part to the project

1. Switch back to the Visual Studio 2010 Lab09 project.
2. In the Solution Explorer window, right-click **Lab09** and click **Add** and then click **New Item**.
3. In the **Add New Item** dialog, in the list of installed templates, click **Visual Web Part**.
4. In the **Name** textbox, type **SkillsManager** and click **Add**.

   The Visual Web Part is created and added to the project.

▶ Task 2: Add a TreeView control to the Visual Web Part

1. Add the following markup to the end of the SkillsManagerUserControl.ascx file in the Visual Web Part:

```
<asp:TreeView ID="skillsMap" runat="server" ShowLines="true">
</asp:TreeView>
```

2. On the **View** menu, click **Code**.

▶ Task 3: Add code for the TreeView control

1. Locate the **using** statements at the top of the code file and add a using statement for **Microsoft.SharePoint**. Your using statements should look as follows:

```
using System;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using Microsoft.SharePoint;
```

2. In the **Page_Load** method, add a **try/catch** block. Your code should look as follows:

```
namespace Lab09.SkillsManager
{
  public partial class SkillsManagerUserControl : UserControl
  {
    protected void Page_Load(object sender, EventArgs e)
    {
      try
      {

      }
      catch (Exception ex)
      {

      }
```

```
    }
  }
}
```

3. In the **try** block, add the following code:

```
skillsMap.Nodes.Clear();
SPWeb thisWeb = SPContext.Current.Web;
SPWeb skillsWeb = thisWeb.Webs["Skills"];
SPList jobs = skillsWeb.Lists["Jobs"];
SPList mashup = skillsWeb.Lists["Mashup"];
SPList skills = skillsWeb.Lists["Skills"];
foreach (SPListItem item in jobs.Items)
{
  TreeNode job = new TreeNode(item["Title"].ToString() + ": "
    + item["Level"].ToString());
  int jobID = item.ID;
  SPQuery skillsMapping = new SPQuery();
  skillsMapping.ViewFields = "<FieldRef Name='LookupSkills' />";
  skillsMapping.Query = "<Where><Eq><FieldRef Name='LookupJobs' />"
    + "<Value Type='Integer'>" + jobID.ToString()
    + "</Value></Eq></Where>";
  SPListItemCollection mappings = mashup.GetItems(skillsMapping);
  if (mappings.Count > 0)
  {
    SPQuery assignedSkills = new SPQuery();
    assignedSkills.ViewFields = "<FieldRef Name='ID' />"
      + "<FieldRef Name='Title' /><FieldRef Name='Importance' />";
    SPListItemCollection jobSkills = skills.GetItems(assignedSkills);
    foreach (SPListItem jobSkill in jobSkills)
    {
      foreach (SPListItem mapping in mappings)
      {
        if(mapping["LookupSkills"].ToString()==jobSkill.ID.ToString())
        {
          job.ChildNodes.Add(new TreeNode(jobSkill["Title"].ToString()
            + ": " + jobSkill["Importance"].ToString()));
        }
      }
    }
  }
  skillsMap.Nodes.Add(job);
}
skillsMap.ExpandAll();
```

4. In the **catch** block, add the following code:

```
catch (Exception ex)
{
  skillsMap.Nodes.Add(new TreeNode(ex.Message));
}
```

▶ Task 4: Add HTML markup to the SkillManager Visual Web Part

1. In the Solution Explorer window, expand the **SkillsManager** node, and double-click **SkillsManagerUserControl.ascx**.
2. Click the **Source** tab.
3. After the closing tag for the **TreeView** element, add the following markup:

```
<br/><br/>
```

4. At the end of the existing markup, add the following HTML link code:

```
<a href="javascript:showSkillAdder();" id="linkAdder"
```

```
  style="display:none;">
  &raquo;Add Skill Mapping
</a>
```

5. At the end of the existing markup, add the following containing a table:

```
<div id="divSkillsManager" style="display:none; padding:5px">
  <table cellpadding='3' cellspacing='3'>
    <tr>
      <td align='center' colspan='2'><b>Skill Mapper</b><br/></td>
    </tr>
    <tr>
      <td>Select a Job:</td>
      <td><select id="Jobs" name="Jobs" /></td>
    </tr>
    <tr>
      <td>Select a Skill:</td>
      <td><select id="Skills" name="Skills" /></td>
    </tr>
    <tr>
      <td colspan='2' align='right'>
        <input type="button" value="Add Skill Mapping"
          style='width:125px;' onclick="addSkillMapping()" />
        <br />
        <input type="button" value="Cancel" style='width:125px'
          onclick="onCancel()" />
      </td>
    </tr>
  </table>
</div>
```

6. At the end of the existing markup, add the following SharePoint script link:

```
<SharePoint:ScriptLink ID="showDialog" runat="server" Name="sp.js"
Localizable="false" LoadAfterUI="true" />
```

7. At the end of the existing markup, add the following JavaScript code:

```
<script language="ecmascript" type="text/ecmascript">

</script>
```

8. Between the begin and end Script tags, add the following variables:

```
var thisDialog;
var jobSelector;
var skillSelector;
var clientCtx;
var skillsWeb;
var mashupList;
var jobList;
var skillList;
var jobItems;
var skillItems;
var mashupItems;
var camlQuery;
var listItemEnumerator;
var listItem;
var selectedJob;
var selectedSkill;
```

9. After the JavaScript variable definition, and before the closing </script> tag, add the following initialization code:

```
_spBodyOnLoadFunctionNames.push("Initialize()");
```

10. After the Initialization code, and before the closing </script> tag, add the following **Initialize()** function:

```
function Initialize() {
  clientCtx = new SP.ClientContext('/skills');
  skillsWeb = clientCtx.get_web();
  jobList = skillsWeb.get_lists().getByTitle("Jobs");
  skillList = skillsWeb.get_lists().getByTitle("Skills");
  mashupList = skillsWeb.get_lists().getByTitle("Mashup");
  camlQuery = new SP.CamlQuery();
  camlQuery.set_viewXml('<View><RowLimit>50</RowLimit></View>');
  jobItems = jobList.getItems(camlQuery);
  skillItems = skillList.getItems(camlQuery);
  clientCtx.load(skillsWeb);
  clientCtx.load(jobList);
  clientCtx.load(skillList);
  clientCtx.load(jobItems, 'Include(Id, Title, Level)');
  clientCtx.load(skillItems, 'Include(Id, Title, Importance)');
  clientCtx.executeQueryAsync(onSkillsWebLoaded,
  errOnSkillsWebLoaded);
}
```

11. After the **Initialize()** function, and before the closing </script> tag, add the following **err On Skills Web Loaded()** function:

```
function errOnSkillsWebLoaded(sender, args) {
  alert(args.get_message());
}
```

12. After the **err On Skills Web Loaded()** function, and before the closing </script> tag, add the following **on Skills Web Loaded()** function:

```
function onSkillsWebLoaded() {
  var linkAdder = document.getElementById("linkAdder");
  linkAdder.style.display = "inline";
}
```

13. After the **on Skills Web Loaded()** function, and before the closing </script> tag, add the following **show Skill Adder()** function:

```
function showSkillAdder() {
  jobSelector = document.getElementById("Jobs");
  skillSelector = document.getElementById("Skills");
  jobSelector.options.length = 0;
  skillSelector.options.length = 0;
  listItemEnumerator = jobItems.getEnumerator();
  while (listItemEnumerator.moveNext()) {
    listItem = listItemEnumerator.get_current();
    jobSelector.options[jobSelector.options.length]
      = new Option(listItem.get_item('Title') + ': '
      + listItem.get_item('Level'), listItem.get_id());
  }
  listItemEnumerator = skillItems.getEnumerator();
  while (listItemEnumerator.moveNext()) {
    listItem = listItemEnumerator.get_current();
    skillSelector.options[skillSelector.options.length]
      = new Option(listItem.get_item('Title') + ': '
      + listItem.get_item('Importance'), listItem.get_id());
  }
  var divSkillsManager = document.getElementById("divSkillsManager");
  divSkillsManager.style.display = "block";
  var dialog = { html: divSkillsManager,
    title: 'Add Skills Mapping', allowMaximize: true,
    showClose: false, width: 400, height: 200};
  thisDialog = SP.UI.ModalDialog.showModalDialog(dialog);
```

```
}
```

14. After the **show Skill Adder()** function, and before the closing </script> tag, add the following **hideSkillMapper()** function:

```
function hideSkillMapper() {
  thisDialog.close();
  window.location.reload(true);
}
```

15. After the **hides kill Mapper ()** function, and before the closing </script> tag, add the following **addSkillMapping()** function:

```
function addSkillMapping() {
  selectedJob =
    jobSelector.options[jobSelector.selectedIndex].value;
  selectedSkill =
    skillSelector.options[skillSelector.selectedIndex].value;
  clientCtx.load(mashupList);
  camlQuery = new SP.CamlQuery();
  camlQuery.set_viewXml('<View><Query><Where><Eq>' +
    '<FieldRef Name=\'LookupJobs\'/><Value Type=\'Integer\'>'
    + selectedJob + '</Value>'
    + '</Eq></Where></Query><RowLimit>50</RowLimit></View>');
  mashupItems = mashupList.getItems(camlQuery);
  clientCtx.load(mashupItems);
  clientCtx.executeQueryAsync(onCurrentMapLoaded,
    errOnCurrentMapLoaded);
}
```

16. After the **add Skill Mapping()** function, and before the closing </script> tag, add the following **onCurrentMapLoaded()** function:

```
function onCurrentMapLoaded() {
  var alreadySelected = false;
  listItemEnumerator = mashupItems.getEnumerator();
  while (listItemEnumerator.moveNext()) {
    listItem = listItemEnumerator.get_current();
    if (listItem.get_item('LookupSkills') == selectedSkill) {
      alreadySelected = true;
      break;
    }
  }
  if (!alreadySelected) {
    var listItemInfo = new SP.ListItemCreationInformation();
    var newListItem = mashupList.addItem(listItemInfo);
    newListItem.set_item('LookupJobs', selectedJob);
    newListItem.set_item('LookupSkills', selectedSkill);
    newListItem.update();
    clientCtx.executeQueryAsync(onMappingAdded, errOnMappingAdded);
    return;
  }
  hideSkillMapper();
}
```

17. After the **on Current Map Loaded()** function, and before the closing </script> tag, add the following **errOnCurrentMapLoaded()** function:

```
function errOnCurrentMapLoaded(sender, args) {
  alert(args.get_message());
}
```

18. After the **err On Current Map Loaded()** function, and before the closing </script> tag, add the following **onMappingAdded()** function:

```
function onMappingAdded(sender, args) {
  hideSkillMapper();
}
```

19. After the **on Mapping Added()** function, and before the closing </script> tag, add the following **errOnMappingAdded()** function:

```
function errOnMappingAdded(sender, args) {
  alert(args.get_message());
}
```

20. After the **err On Mapping Added()** function, and before the closing </script> tag, add the following **onSkillMapped()** function:

```
function onSkillMapped() {
  hideSkillMapper();
}
```

21. After the **on Skill Mapped()** function, and before the closing </script> tag, add the following **onCancel()** function:

```
function onCancel() {
  hideSkillMapper();
}
```

22. On the **File** menu, click **Save All**.

Task 5: Deploy and test the client-side dialog

1. In the Solution Explorer window, right-click the **Lab09** project and click **Deploy**.
2. Start **Internet Explorer** and browse to **http://sharepoint**.
3. On the **Site Actions** menu, click **New Page**.
4. In the **New page name** text box, type **Skills** and then click **Create**.
5. On the ribbon, click the **Insert** tab.
6. On the ribbon, click **Web Part**.
7. In the **Categories** section, click **Custom**.
8. In the **Web Parts** section, click **Skills Manager**.
9. Click **Add**.

   The Web Part is added to the page.

10. On the ribbon, click **Save and Close**.

    Note the new Add Skill Mapping HTML link that appears in the Visual Web Part.

11. Click **Add Skill Mapping**.
12. In the dialog, in the **Select a job** drop-down list, click **Developer: Senior**.
13. In the **Select a Skill** drop-down list, click **Make Technical Decisions: Preferred**.
14. Click **Add Skill Mapping**.

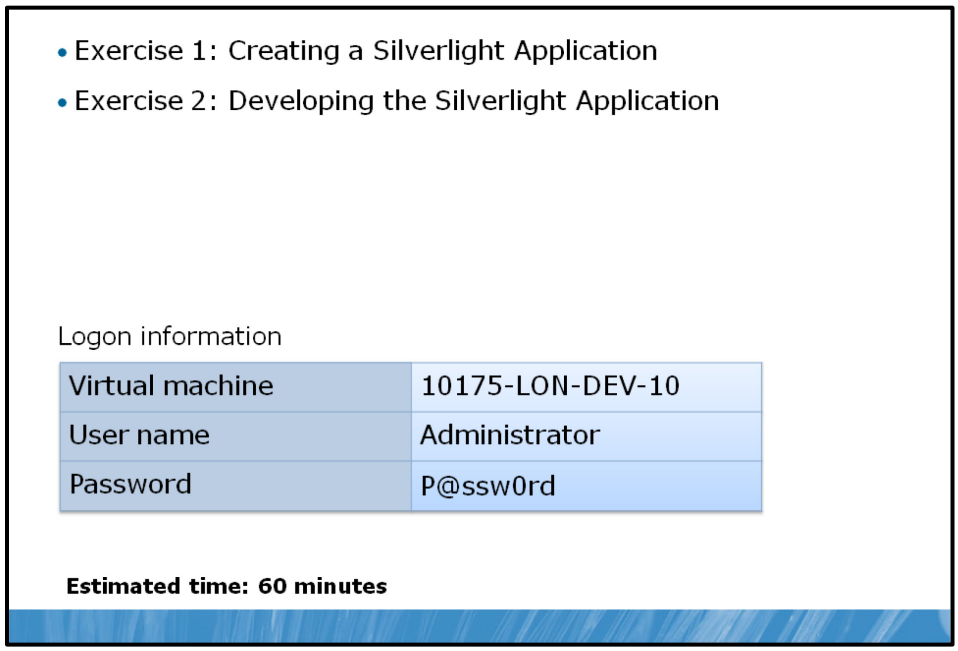
Results: After this exercise, you should have created a SharePoint page that uses the JavaScript client object model to manipulate lists and sites on the server.
This page uses the hidden DIV dialog approach for modal dialogs in the SharePoint site.

Module 10

Lab Instructions: Developing Silverlight Applications for SharePoint

Contents:

Lab: Developing Silverlight Applications by Using the SharePoint Client Object Model



Log On to the Virtual Machine for This Lab

For this lab, you use the available virtual machine environment. Before you begin the lab, you must log on to the **10175-LON-DEV-10** virtual machine as **Administrator** using the password **P@ssw0rd**.

Exercise 1: Creating a Silverlight Application

Scenario

In this exercise, you create a Silverlight application that retrieves information from the SharePoint site and populates a drop-down list of document libraries.

The main tasks for this exercise are as follows:

1. Create a Silverlight project.
2. Prepare the Silverlight Extensible Application Markup Language (XAML) interface.

▶ Task 1: Create a Silverlight project

1. On the **Start** menu, click **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.

   Microsoft Visual Studio® opens.

2. On the **File** menu, point to **New**, and then click **Project**.
3. In the **Installed Templates** section, expand the **Visual C#** node, and then click **Silverlight**.
4. Click **Silverlight Application**.
5. In the **Name** text box, type **Lab10**.
6. In the **Location** text box, type **E:\Student\Lab10\Starter**, and then click **OK**.
7. In the **New Silverlight Application** dialog box, clear the **Host the Silverlight application in a new Web site** check box, and then click **OK**.

   The Microsoft Silverlight project is created.

▶ Task 2: Prepare the Silverlight user interface

1. In the **UserControl** element, after the **d:DesignWidth="400"** attribute and before the closing Extensible Markup Language (XML) tag, add the following attributes:

```
Height="430" Width="600"
```

2. In the **MainPage.xaml** file, in the **xaml** view, add the following code between the open and close **<Grid>** tags:

```
<ProgressBar Height="20" HorizontalAlignment="Left"
  Background="Black" Margin="10,10,10,0" Name="Loader"
  VerticalAlignment="Top" Width="580" />
<TextBlock Height="20" HorizontalAlignment="Left"
  Margin="10,35,0,0" Name="Status" Text="Status"
  VerticalAlignment="Top"/>
<ComboBox Name="LibraryPicker" Height="30" Width="500"
  HorizontalAlignment="Left" Margin="10,60,0,0"
  VerticalAlignment="Top"/>
<StackPanel Orientation="Horizontal" Margin="10,100,0,0"
  Height="320">
<ScrollViewer Width="235" Height="310"  HorizontalAlignment="Left"
  VerticalAlignment="Top" HorizontalContentAlignment="Left"
  VerticalContentAlignment="Top">
  <StackPanel  Width="210"
    ScrollViewer.VerticalScrollBarVisibility="Auto"
    ScrollViewer.HorizontalScrollBarVisibility="Auto"
    Name="myContainer" HorizontalAlignment="Left"
    VerticalAlignment="Top"  />
</ScrollViewer>
```

```
<StackPanel Name="myMedia" Height="310" Width="340"
  Margin="10,0,0,0" HorizontalAlignment="Left"
  VerticalAlignment="Top"  />
</StackPanel>
```

3. On the **File** menu, click **Save All**.

Exercise 2: Developing the Silverlight Application

Scenario

In this exercise, you develop Silverlight application so that when the user clicks a library, the images or .wmv files are retrieved from the library and displayed in the Silverlight panel.

When clicked, an item is retrieved and shown to the right of the thumbnail images. If the item is a video, it is played automatically.

The main tasks for this exercise are as follows:

1. Set the build location for the Silverlight application.
2. Add references to the client object model DLL.
3. Add **using** statements to the **App.xaml.cs** file.
4. Add the **ApplicationContext** to the **Application_Startup** method.
5. Add using statements to **MainPage.xaml.cs**.
6. Add variables to **MainPage.xaml.cs**.
7. Add the **Loaded Event** to the **MainPage**.
8. Add event methods to **MainPage.xaml.cs**.
9. Build and test the application.

▶ Task 1: Set the build location

1. In the Solution Explorer window, right-click **Lab10**, and then click **Properties**.
2. In the **Tabs** section, on the left side of the **Properties** page, click **Build**.
3. In the **Output path** section, click **Browse**, and then browse to the following location: **C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions \14\TEMPLATE\LAYOUTS\ClientBin**
4. In the Select Output Path window, click **Select Folder**.
5. On the **File** menu, click **Save All**.
6. Close the **Properties** tab.

▶ Task 2: Add a reference to the Microsoft SharePoint client object model DLLs

1. In the Solution Explorer window, right-click the **References** node, and click **Add Reference**.
2. In the **Add Reference** dialog box, click the **Browse** tab, and then browse to the following location: **C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions \14\Template\Layouts\ClientBin**
3. Click **Microsoft.SharePoint.Client.Silverlight.dll**, hold down the CTRL key, click **Microsoft.SharePoint.Client.Silverlight.Runtime.dll**, and then click **OK**.

▶ Task 3: Add using statements to App.xaml.cs

1. In the Solution Explorer window, expand the **App.xaml** node, and double-click the **App.xaml.cs** file.
2. At the end of the current **using** statements, add the following code:

```
using Microsoft.SharePoint.Client;
using System.Threading;
```

Task 4: Add the ApplicationContext to the Application_Startup method

1. In the **Application_Startup** method, before the existing line of code, add the following code:

```
ApplicationContext.Init(
  e.InitParams, SynchronizationContext.Current);
```

2. On the **File** menu, click **Save All**.
3. Close the **App.xaml.cs** file.

Task 5: Add using statements to MainPage.xaml.cs

1. In the Solution Explorer window, expand the **MainPage.xaml** node, and double-click the **MainPage.xaml.cs** file.
2. At the end of the current **using** statements, add the following code:

```
using Microsoft.SharePoint.Client;
using System.Threading;
using System.Windows.Media.Imaging;
```

Task 6: Add variables to MainPage.xaml.cs

1. In the **MainPage.xaml.cs** file, in the class **MainPage**, add the following code:

```
ClientContext clientCtx;
Microsoft.SharePoint.Client.List mediaLib;
int fileTracker = 0;
int rowTracker = 0;
StackPanel row = new StackPanel();
```

2. On the **File** menu, click **Save All**.

Task 7: Add the Loaded event to MainPage

1. In the Solution Explorer window, open the **MainPage.xaml**.
2. In the **<Grid>** element, after the background element and before the closing XML tag, add the following code:

```
Loaded="LayoutRoot_Loaded"
```

3. On the **File** menu, click **Save All**.
4. On the **View** menu, click **Code**.

Task 8: Add event methods to MainPage.xaml.cs

1. After the **MainPage()** method, add the following code:

```
private void LayoutRoot_Loaded(object sender, RoutedEventArgs e)
{
  Loader.Maximum = 2;
  Loader.Value = 0;
  Status.Text = "Connecting to Web...";
  clientCtx = new ClientContext(ApplicationContext.Current.Url);
  clientCtx.Load(clientCtx.Web);
  clientCtx.ExecuteQueryAsync(updateConnectionStatus,
    errUpdateConnectionStatus);
}
```

2. After the **LayoutRoot_Loaded()** method, add the following **updateConnectionStatus()** method:

```
void updateConnectionStatus(Object sender,
  ClientRequestSucceededEventArgs e)
{
```

```
  Dispatcher.BeginInvoke(makeProgressWebConnection);
}
```

3. After the **updateConnectionStatus()** method, add the following **errUpdateConnectionStatus()** method:

```
void errUpdateConnectionStatus(Object sender, ClientRequestFailedEventArgs e)
{
  Status.Text = e.Message;
}
```

4. After the **errUpdateConnectionStatus()** method, add the following **makeProgressWebConnection()** method:

```
void makeProgressWebConnection()
{
  Loader.Value++;
  Status.Text = "Web Connected. Connecting to media stores...";
  clientCtx.Load(clientCtx.Web.Lists, lists => lists
    .Include(list => list.Title, list => list.Id)
    .Where(list => list.BaseType==BaseType.DocumentLibrary
      && !list.Hidden));
  clientCtx.ExecuteQueryAsync(updateLists, errUpdateLists);
}
```

5. After the **makeProgressWebConnection()** method, add the following **updateLists()** method:

```
void updateLists(Object sender, ClientRequestSucceededEventArgs e)
{
  Dispatcher.BeginInvoke(makeProgressListConnection);
}
```

6. After the **updateLists()** method, add the following **errUpdateLists()** method:

```
void errUpdateLists(Object sender, ClientRequestFailedEventArgs e)
{
  Status.Text = e.Message;
}
```

7. After the **errUpdateLists ()** method, add the following **makeProgressListConnection()** method:

```
void makeProgressListConnection()
{
  Loader.Value++;
  Status.Text = "Now showing all media stores";
  foreach (List mediaStore in clientCtx.Web.Lists)
  {
    ListBoxItem mediaLibPicker = new ListBoxItem();
    mediaLibPicker.Content = mediaStore.Title;
    mediaLibPicker.Tag = mediaStore.Id;
    mediaLibPicker.MouseLeftButtonUp += new
      MouseButtonEventHandler(mediaLibPicker_MouseLeftButtonUp);
    LibraryPicker.Items.Add(mediaLibPicker);
  }
}
```

8. After the **makeProgressListConnection()** method, add the following **mediaLibPicker_MouseLeftButtonUp()** method:

```
void mediaLibPicker_MouseLeftButtonUp(object sender, MouseButtonEventArgs e)
{
  ListBoxItem src = (ListBoxItem)sender;
  string libID = src.Tag.ToString();
  mediaLib = clientCtx.Web.Lists.GetById(new Guid(libID));
  clientCtx.Load(mediaLib);
  clientCtx.Load(mediaLib.RootFolder);
```

```
  clientCtx.Load(mediaLib.RootFolder.Files);
  clientCtx.ExecuteQueryAsync(getListData, errGetListData);
}
```

9. After the **mediaLibPicker_MouseLeftButtonUp()** method, add the following **getListData()** method:

```
void getListData(Object sender, ClientRequestSucceededEventArgs e)
{
  Dispatcher.BeginInvoke(loadFiles);
}
```

10. After the **getListData()** method, add the following **errGetListData()** method:

```
void errGetListData(Object sender, ClientRequestFailedEventArgs e)
{
  Status.Text = e.Message;
}
```

11. After the **errGetListData()** method, add the following **loadFiles ()** method:

```
private void loadFiles()
{
  myContainer.Children.Clear();
  Loader.Maximum = mediaLib.RootFolder.Files.Count;
  if (Loader.Maximum != 0)
  {
    Loader.Value = 0;
    Status.Text = "Loading Files...";
    fileTracker = 0;
    foreach (File fle in mediaLib.RootFolder.Files)
    {
      clientCtx.Load(fle);
      clientCtx.ExecuteQueryAsync(addFileToUI, errAddFileToUI);
    }
    return;
  }
  Status.Text = "There are no files to show from this library";
}
```

12. After the **loadFiles()** method, add the following **addFileToUI()** method:

```
void addFileToUI(Object sender, ClientRequestSucceededEventArgs e)
{
  Dispatcher.BeginInvoke(addFile);
}
```

13. After the **addFileToUI()** method, add the following **errAddFileToUI()** method:

```
void errAddFileToUI(Object sender, ClientRequestFailedEventArgs e)
{
  Status.Text = e.Message;
}
```

14. After the **errAddFileToUI()** method, add the following **addFile()** method:

```
void addFile()
{
  string fName = mediaLib.RootFolder.Files[fileTracker].Name;
  string fUrl =
    mediaLib.RootFolder.Files[fileTracker].ServerRelativeUrl;
  fUrl = clientCtx.Url + fUrl;
  if ((rowTracker % 2) == 0)
  {
    row = new StackPanel();
    row.Orientation = Orientation.Horizontal;
    myContainer.Children.Add(row);
  }
```

```
  if ((fName.ToLower().EndsWith(".png")) ||
    (fName.ToLower().EndsWith(".jpeg")) ||
    (fName.ToLower().EndsWith(".jpg")))
  {
    Image img = new Image();
    img.MaxWidth = 100;
    img.MaxHeight = 50;
    img.Stretch = Stretch.Uniform;
    BitmapImage bitMap = new BitmapImage(
      new Uri(fUrl, UriKind.Absolute));
    img.Source = bitMap;
    ContentControl mediaButton = new ContentControl();
    mediaButton.Margin = new Thickness(2.5);
    mediaButton.Tag = fUrl;
    ToolTipService.SetToolTip(mediaButton, fName);
    mediaButton.Cursor = Cursors.Hand;
    mediaButton.MouseLeftButtonUp += new
      MouseButtonEventHandler(mediaButton_MouseLeftButtonUp);
    mediaButton.Content = img;
    row.Children.Add(mediaButton);
    rowTracker++;
  }
  if (fName.ToLower().EndsWith(".wmv"))
  {
    MediaElement media = new MediaElement();
    media.MaxWidth = 100;
    media.MaxHeight = 50;
    media.Stretch = Stretch.Uniform;
    media.Source = new Uri(fUrl, UriKind.Absolute);
    media.AutoPlay = false;
    ContentControl mediaButton = new ContentControl();
    mediaButton.Margin = new Thickness(2.5);
    mediaButton.Tag = fUrl;
    ToolTipService.SetToolTip(mediaButton, fName);
    mediaButton.Cursor = Cursors.Hand;
    mediaButton.MouseLeftButtonUp += new
      MouseButtonEventHandler(mediaButton_MouseLeftButtonUp);
    mediaButton.Content = media;
    row.Children.Add(mediaButton);
    rowTracker++;
  }
  Loader.Value++;
  fileTracker++;
  if (fileTracker >= mediaLib.RootFolder.Files.Count)
  {
    Status.Text = "All files have now been loaded";
  }
}
```

15. After the **addFile()** method, add the following **mediaButton_MouseLeftButtonUp()** method:

```
void mediaButton_MouseLeftButtonUp(object sender, MouseButtonEventArgs e)
{
  myMedia.Children.Clear();
  ContentControl src = (ContentControl)sender;
  Uri fUri = new Uri(src.Tag.ToString(), UriKind.Absolute);
  if ((fUri.OriginalString.EndsWith(".png"))
    || (fUri.OriginalString.EndsWith(".jpg")))
  {
    Image img = new Image();
    img.MaxWidth = 340;
    img.MaxHeight = 310;
    img.Stretch = Stretch.Uniform;
    BitmapImage bitMap = new BitmapImage(fUri);
    img.Source = bitMap;
```

```
    myMedia.Children.Add(img);
    return;
  }
  if (fUri.OriginalString.EndsWith(".wmv"))
  {
    MediaElement media = new MediaElement();
    media.MaxWidth = 340;
    media.MaxHeight = 310;
    media.Stretch = Stretch.Uniform;
    media.Source = fUri;
    media.AutoPlay = true;
    myMedia.Children.Add(media);
  }
```

16. On the **File** menu, click **Save All**.

Task 9: Build and test the Silverlight application

1. In the Solution Explorer window, right-click **Lab10**, and click **Build**.
2. Switch back to Windows® Internet Explorer® and browse to the site **http://sharepoint**.
3. On the **Quick Launch** bar, click **Site Pages** and then click **Training**.
4. On the ribbon, click **Edit**.
5. On the ribbon, click the **Insert** tab.
6. On the ribbon, in the **Web Parts** section, click **Web Part**.
7. In the **Categories** section, in the **Media and Content** category, click **Silverlight Web Part**, and then click **Add**.
8. In the **Silverlight Web Part URL**, type **/_layouts/ClientBin/Lab10.xap**, and then click **OK**.
9. On the **Web Part** drop-down menu, click **Edit Web Part**.
10. On the right side of the screen, in the Silverlight Web Part properties panel, in the **Height** section, click the **Yes** option, and type **600** in the text box.
11. In the **Width** section, click **No, Adjust width to fit zone**, and then click **OK**.
12. On the ribbon, click **Save and Close**.
13. In the drop-down list in the Silverlight application, click **Shared Documents**.

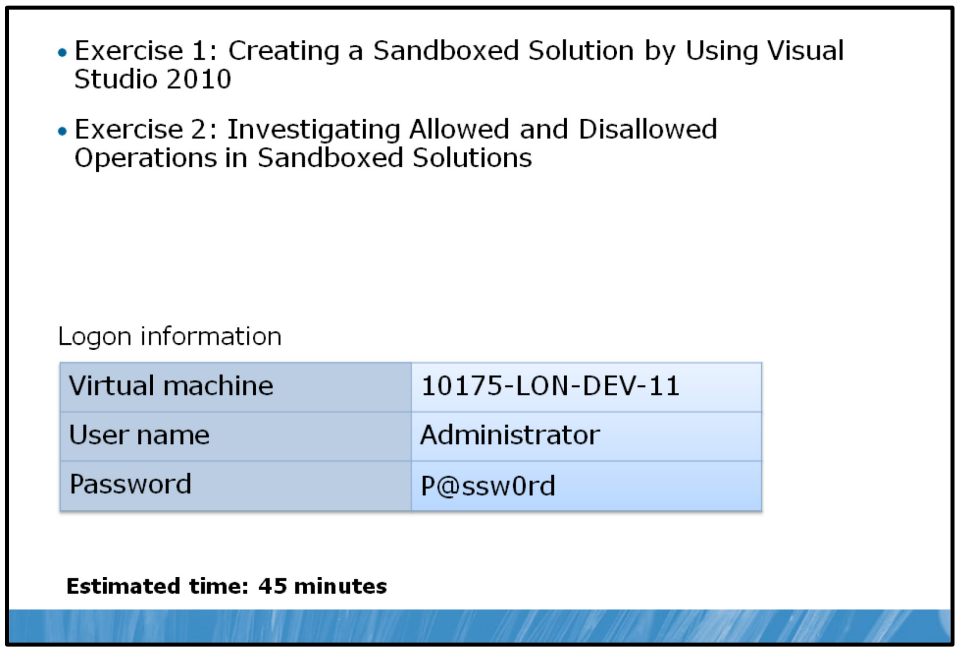
    After a few seconds, video thumbnails are displayed.
14. Click each video thumbnail and watch a few seconds of each video.
15. Close all applications. You have now completed this lab.

Module 11

Lab Instructions: Developing Sandboxed Solutions

Contents:

Lab: Creating Sandboxed Solutions for SharePoint 2010



Log On to the Virtual Machine for This Lab

For this lab, you use the available virtual machine environment. Before you begin the lab, you must log on to the **10175-LON-DEV-11** virtual machine as **Administrator**, with the password **P@ssw0rd**.

Exercise 1: Creating a Sandboxed Solution by Using Visual Studio 2010

Task 1: Create an Empty SharePoint Project

1. On the **Start** menu, click **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.
2. On the **File** menu, point to **New**, and then click **Project**.
3. In the **Installed Templates** section, expand the **Visual C#** node, expand the **SharePoint** node and then click **2010**.
4. Click **Empty SharePoint Project**.
5. In the **Name** text box, type **Lab11**.
6. In the **Location** text box, type **E:\Student\Lab11\Starter** and click **OK**.

   The SharePoint Customization Wizard appears.
7. In the **What local site do you want to use for debugging?** text box, type http://sharepoint.
8. In the **What is the trust level for this SharePoint solution** section, click the **Deploy as a sandboxed solution** option.
9. Click **Finish**.

   The project is created.

Task 2: Add Artifacts to the project

1. In the Solution Explorer window, right-click **Lab11**, click **Add**, and then click **New Item**.
2. Click **Visual Web Part**.
3. In the **Name** text box, type **TestSB_VWP** and then click **Add**.
4. In the Solution Explorer window, right-click **Lab11**, click **Add**, and then click **New Item**.
5. Click **Application Page**.
6. In the **Name** text box, type **TestSB_AppPage.aspx** and then click **Add**.
7. In the Solution Explorer window, right-click **Lab11** and then click **Deploy**.
   Note that the project is not built and deployed because user controls and application pages are not allowed in sandboxed solutions.
8. In the Solution Explorer window, right-click **TestSB_VWP**, click **Delete**, and then click **OK**.
9. In the Solution Explorer window, right-click **Layouts**, click **Delete**, and then click **OK**.

Task 3: Add a Web Part to the project

1. In the Solution Explorer window, right-click **Lab11**, click **Add**, and then click **New Item**.
2. Click **Web Part**.
3. In the **Name** text box, type **BonnevilleTestBed** and then click **Add**.
4. Add the following code to the empty line above the namespace declaration:

```
using Microsoft.SharePoint.Administration;
```

5. Place the cursor after the opening brace of the **public class BonnevilleTestBed** class declaration and press ENTER to create a new line.
6. In the space you have just created, add the following code:

```
LiteralControl output = new LiteralControl();
```

7. Add the following code to the **CreateChildControls** method:

```
SPFarm thisFarm = SPFarm.Local;
output.Text = thisFarm.Name;
this.Controls.Add(output);
```

8. On the **File** menu, click **Save All**.

Task 4: Deploy and test the sandboxed solution

1. In the Solution Explorer window, right-click **Lab11** and then click **Deploy**.
2. Use Windows® Internet Explorer® to browse to **http://sharepoint**.
3. On the **Site Actions** menu, click **Site Settings**.
4. In the **Galleries** section, click **Solutions**.

   The sandboxed solution you have just deployed is listed.
5. In the breadcrumb control, click **Home**.
6. On the ribbon, click **Edit**.
7. On the ribbon, click the **Insert** tab.
8. On the ribbon, click **Web Part**.
9. In the **Categories** section, click **Custom**.
10. In the **Web Parts** section, click **BonnevilleTestBed**.
11. Click **Add**.
12. On the ribbon, click **Save & Close**.
13. Read the error that is displayed in the Web Part. This error is displayed because code in the Web Part accesses the prohibited **SPFarm** object.
14. On the **Site Actions** menu, click **Site Settings**.
15. In the **Galleries** section, click **Solutions**.
16. Point to **Lab11** and then click the drop-down arrow that appears.
17. Click **Deactivate**.
18. In the **Solution Gallery - Deactivate Solution** dialog box, click **Deactivate**.
19. In the breadcrumb control, click **Home**.
20. Read the error that is displayed in the Web Part. This error is displayed because the solution has been deactivated.
21. Leave Internet Explorer running.

Results: After this exercise, you should have verified that sandboxed solutions cannot contain some specific artifacts, and that some objects cannot be used in sandboxed solutions.

Exercise 2: Investigating Allowed and Disallowed Operations in Sandboxed Solutions

▶ Task 1: Complete the Web Part

1. Switch to Visual Studio.
2. Select the three lines of code you previously added to the **CreateChildControls** method.
3. On the toolbar, click **Comment out the selected lines**.
4. Add the following function after the closing brace of the **CreateChildControls** method:

```
void renderWebInfo_Click(object sender, EventArgs e)
{
  try
  {
    SPWeb thisWeb = SPContext.Current.Web;
    string message = string.Format(
      "This web contains {0} subwebs", thisWeb.Webs.Count);
    output.Text = message;
  }
  catch (Exception ex)
  {
    output.Text = ex.Message;
  }
}
```

5. Add the following functions after the function you have just added:

```
void renderWebInfoElevated_Click(object sender, EventArgs e)
{
  try
  {
    SPSecurity.RunWithElevatedPrivileges(showWebCount);
  }
  catch (Exception ex)
  {
    output.Text = "Error caught by my code: " + ex.Message;
  }
}

void showWebCount()
{
  SPWeb thisWeb = SPContext.Current.Web;
  string message = string.Format(
    "This web contains {0} subwebs", thisWeb.Webs.Count);
  output.Text = message;
}
```

6. Add the following function after the functions you have just added:

```
void accessProhibitedNamespace_Click(object sender, EventArgs e)
{
  try
  {
    System.Net.HttpWebRequest.Create(
      "http://intranet.contoso.com");
    output.Text = "Success";
  }
  catch (System.Security.SecurityException secEx)
  {
    output.Text = "Security Violation! Error caught by my code: "
      + secEx.Message;
  }
  catch (Exception ex)
```

```
  {
    output.Text = "Generic Exception. Error caught by my code: "
      + ex.Message;
  }
}
```

7. Add the following code to the **CreateChildControls** method:

```
Button renderWebInfo = new Button();
renderWebInfo.Text = "Access data in this site";
renderWebInfo.Click += new EventHandler(renderWebInfo_Click);
Button renderWebInfoElevated = new Button();
renderWebInfoElevated.Text = "Use elevated privileges";
renderWebInfoElevated.Click
  += new EventHandler(renderWebInfoElevated_Click);
Button accessProhibitedNamespace = new Button();
accessProhibitedNamespace.Text = "Create an HTTP connection ";
accessProhibitedNamespace.Click
  += new EventHandler(accessProhibitedNamespace_Click);
this.Controls.Add(output);
this.Controls.Add(new LiteralControl("<br />"));
this.Controls.Add(renderWebInfo);
this.Controls.Add(renderWebInfoElevated);
this.Controls.Add(accessProhibitedNamespace);
```

8. On the **File** menu, click **Save All**.

▶ Task 2: Deploy and test the sandboxed Web Part

1. In the Solution Explorer window, right-click **Lab11** and then click **Deploy**.
2. Switch to Internet Explorer.
3. On the **Site Actions** menu, click **Site Settings**.
4. In the **Galleries** section, click **Solutions**.

   The Lab 11 solution has been re-deployed and activated by Visual Studio.
5. In the breadcrumb control, click **Home**.

   Three buttons are displayed in the Web Part.
6. Click **Access data in this site**.
7. Read the text displayed in the label above the button. The code has run successfully in the sandboxed Web Part.
8. Click **Use Elevated Privileges**.

   Read the error that is displayed in the Web Part. This error is displayed because code in the Web Part has attempted to perform a prohibited operation by running code with elevated privileges.
9. In the breadcrumb control, click **Home** to refresh the page.
10. Click **Create an HTTP connection**.
11. Read the error that is displayed in the Web Part. Note that this error message has been provided by the code you added, so your exception handler has caught this security violation in a sandboxed Web Part.
12. Close all applications. You have now completed this lab.

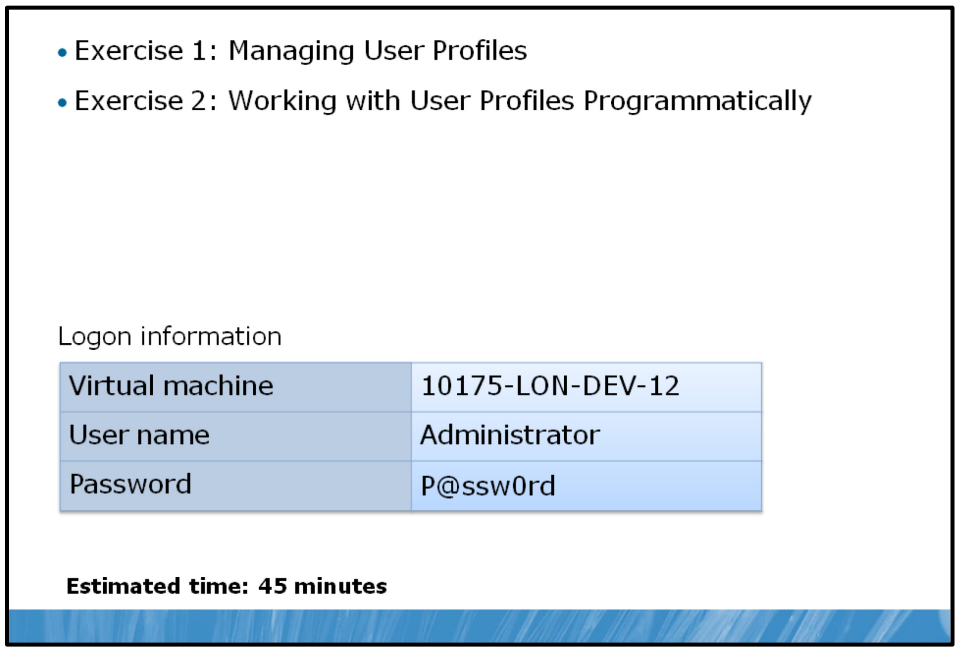
Results: After this exercise, you should have verified that some operations are not allowed in a sandboxed solution.

Module 12

Lab Instructions: Working with SharePoint Server Profiles and Taxonomy APIs

Contents:

Lab: Working with User Profiles and Taxonomies Programmatically



Log On to the Virtual Machine for This Lab

For this lab, you use the available virtual machine environment. Before you begin the lab, you must log on to the **10175-LON-DEV-12** virtual machine as **Administrator** using the password **P@ssw0rd**.

Exercise 1: Managing User Profiles

Task 1: Manage user profile properties

1. On the **Start** menu, click **All Programs**, click **Microsoft SharePoint 2010 Products**, and then click **SharePoint 2010 Central Administration**.
2. In the **Application Management** section, click **Manage service applications**.
3. Click **User Profile Service Application**.
4. In the **People** section, click **Manage User Properties**.
5. In the **Contact Information** section, point to **Work e-mail**, and then click the drop-down arrow that appears.
6. Click **Edit**.
7. In the **Edit Settings** section, click **Allow users to edit values for this property**.
8. In the **Display Settings** section, select the **Show in the profile properties section of the user's profile page** check box, and then click **OK**.
9. Close Windows® Internet Explorer®.

Task 2: Create user profiles

1. Start Internet Explorer, and browse to **http://sharepoint**.
2. In the top right section of the page, click the **SHAREPOINT\administrator** drop-down menu, and then click **Sign in as a Different User**.
3. In the **User Name** text box, type **SHAREPOINT\KrishnaS**.
4. In the **Password** text box, type **P@ssw0rd**, and then click **OK**.
5. In the top right section of the page, click the **SHAREPOINT\krishnas** drop-down menu, and then click **My Profile**.
6. Below the picture, click **Edit My Profile**.
7. In the **Work e-mail** text box, type **krishnas@sharepoint.com**.
8. Click **Save and Close**.
9. Near the top of the page, click **My Content**.

   The personal site is created for Krishna.
10. Close Internet Explorer.
11. Start Internet Explorer, and browse to **http://sharepoint**.
12. In the top right section of the page, click the **SHAREPOINT\administrator** drop-down menu, and then click **Sign in as Different User**.
13. In the **User Name** text box, type **SHAREPOINT\MartinR**.
14. In the **Password** text box, type **P@ssw0rd**, and then click **OK**.
15. In the top right section of the page, click the **SHAREPOINT\martinr** drop-down menu, and then click **My Profile**.
16. Below the picture, click **Edit My Profile**.

17. In the **Work e-mail** text box, type **martinr@sharepoint.com**.
18. Click **Save and Close**.
19. Close Internet Explorer.
20. Start Internet Explorer, and browse to **http://sharepoint**.
21. In the top right section of the page, click the **SHAREPOINT\administrator** drop-down menu, and then click **Sign in as Different User**.
22. In the **User Name** text box, type **SHAREPOINT\BennoK**.
23. In the **Password** text box, type **P@ssw0rd**, and then click **OK**.
24. In the top right section of the page, click t**h**e **SHAREPOINT\bennok** drop-down menu, and then click **My Profile**.
25. Below the picture, click **Edit My Profile**.
26. Click **Save and Close**.
27. Close Internet Explorer.
28. Start Internet Explorer, and browse to **http://sharepoint**.
29. In the top right section of the page, click the **SHAREPOINT\administrator** drop-down menu, and then click **Sign in as Different User**.
30. In the **User Name** text box, type **SHAREPOINT\LauraG**.
31. In the **Password** text box, type **P@ssw0rd**, and then click **OK**.
32. In the top right section of the page, click the **SHAREPOINT\laurag** drop-down menu, and then click **My Profile**.
33. Below the picture, click **Edit My Profile**.
34. Click **Save and Close**.
35. Close Internet Explorer.

Results: After this exercise, you should have created various user profiles by logging in as different users.

Exercise 2: Working with User Profiles Programmatically

▶ Task 1: Create an Empty SharePoint Project

1. On the **Start** menu, click **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.

   Microsoft Visual Studio® 2010 opens.

2. On the **File** menu, point to **New**, and then click **Project**.
3. In the **Installed Templates** section, expand the **Visual C#** node, expand the **SharePoint** node, and then click **2010**.
4. Click **Empty SharePoint Project**.
5. In the **Name** text box, type **Lab12**.
6. In the **Location** text box, type **E:\Student\Lab12\Starter**, and then click **OK**.

   The SharePoint Customization Wizard appears.

7. In the **What local site do you want to use for debugging?** text box, type **http://sharepoint**.
8. In the **What is the trust level for this SharePoint solution** section, click the **Deploy as a farm solution** option.
9. Click **Finish**.

   The project is created.

▶ Task 2: Create an application page for working with user profiles

1. In the Solution Explorer window, right-click **Lab12**, point to **Add**, and then click **New Item**.
2. Click **Application Page**.
3. In the **Name** text box, type **ProfileReporter**, and then click **Add**.
4. Add the following markup between the opening and closing tags of the **<asp:Content>** element that has an **ID** of **Main**:

```
<asp:Label ID="Label1" runat="server"
  Text="Profiles and My Sites"></asp:Label>
<asp:Panel ID="configuredMySites" runat="server" BorderColor="Blue"
BorderStyle="Dashed" BorderWidth="1">
  <asp:Label ID="configuredLabel" runat="server"
  Text="Profiles, Emails, and Personal Sites"></asp:Label>
  <br/>
  <asp:Table ID="configuredTable" runat="server">
  </asp:Table>
</asp:Panel>
```

5. On the **View** menu, click **Code**.
6. In the Solution Explorer window, right-click **Lab12**, and then click **Add Reference**.
7. Click the **.NET** tab.
8. Click **Microsoft.Office.Server**, hold down the CTRL key, and click **Microsoft.Office.Server.UserProfiles**. Click **OK**.

▶ Task 3: Create the application page

1. Add the following **using** statements to directly above the namespace declaration:

```
using System.Web.UI;
using System.Web.UI.WebControls;
using Microsoft.Office.Server;
using Microsoft.Office.Server.UserProfiles;
```

2. Add the following code directly above the **Page_Load** method:

```
UserProfileManager uMan=null;
```

3. Add the following code directly after the closing brace of the **Page_Load** method:

```
void emailUpdater_Click(object sender, EventArgs e)
{
  try
  {
    Button src = (Button)sender;
    UserProfile userProfile =
    (UserProfile)uMan.GetUserProfile(long.Parse(src.CommandName));
    string accountName =
      userProfile["AccountName"].Value.ToString();
    int iPos = accountName.IndexOf("\\");
    accountName = accountName.Substring(iPos + 1);
    string updatedEmail = accountName + "@sharepoint.com";
    userProfile["WorkEmail"].Value = updatedEmail;
    userProfile.Commit();
  }
  catch (Exception ex)
  {
    this.Page.Response.Write(ex.Message);
  }
}
```

4. Add the following code directly after the method you have just added:

```
void mySiteCreator_Click(object sender, EventArgs e)
{
  try
  {
    Button src = (Button)sender;
    UserProfile userProfile =
    (UserProfile)uMan.GetUserProfile(long.Parse(src.CommandName));
    userProfile.CreatePersonalSite();
  }
  catch (Exception ex)
  {
    this.Page.Response.Write(ex.Message);
  }
}
```

5. Add the following code to the **Page_Load** method:

```
try
{
  SPServiceContext svrCtx = SPServiceContext.Current;
  uMan = new UserProfileManager(svrCtx);
  TableCell cell;
  TableRow row;
  row = new TableRow();
  cell = new TableCell();
  cell.Text = "User";
  cell.Font.Bold = true;
  row.Cells.Add(cell);
  cell = new TableCell();
  cell.Text = "Email";
  cell.Font.Bold = true;
  row.Cells.Add(cell);
```

```
cell = new TableCell();
cell.Text = "Personal Site";
cell.Font.Bold = true;
row.Cells.Add(cell);
configuredTable.Rows.Add(row);
foreach (UserProfile uProfile in uMan)
{
  ProfileSubtypePropertyManager propMan =
    uMan.DefaultProfileSubtypeProperties;
  string accountName = uProfile["AccountName"].Value.ToString();
  string emailAddr = string.Empty;
  string personalSite = string.Empty;
  if (uProfile["WorkEmail"].Value != null)
  {
    emailAddr = uProfile["WorkEmail"].Value.ToString();
  }
  if (uProfile["PersonalSpace"].Value != null)
  {
    personalSite = uProfile["PersonalSpace"].Value.ToString();
   }
  row = new TableRow();
  cell = new TableCell();
  cell.Text = accountName;
  row.Cells.Add(cell);
  cell = new TableCell();
  if (emailAddr != string.Empty)
  {
    HyperLink emailHyper = new HyperLink();
    emailHyper.Text = emailAddr;
    emailHyper.NavigateUrl = "mailto:" + emailAddr;
    cell.Controls.Add(emailHyper);
  }
  else
  {
    if (uProfile["AccountName"].Value.ToString().ToLower()
      ==SPContext.Current.Web.CurrentUser.LoginName.ToLower())
    {
      Button emailUpdater = new Button();
      emailUpdater.CommandName = uProfile.RecordId.ToString();
      emailUpdater.Text = "Set to default...";
      emailUpdater.Click +=
        new EventHandler(emailUpdater_Click);
      cell.Controls.Add(emailUpdater);
    }
    else
    {
      cell.Text = "Email not set";
    }
  }
  row.Cells.Add(cell);
  cell = new TableCell();
  if ((personalSite == "http://")
    || (personalSite == string.Empty))
  {
    if (uProfile["AccountName"].Value.ToString().ToLower()
      == SPContext.Current.Web.CurrentUser.LoginName.ToLower())
    {
      Button mySiteCreator = new Button();
      mySiteCreator.CommandName = uProfile.RecordId.ToString();
      mySiteCreator.Text = "Create...";
      mySiteCreator.Click
        += new EventHandler(mySiteCreator_Click);
      cell.Controls.Add(mySiteCreator);
    }
```

```
      else
      {
        cell.Text = "Personal site not created";
      }
    }
    else
    {
      cell.Text = personalSite;
    }
    row.Cells.Add(cell);
    configuredTable.Rows.Add(row);
  }
}
catch (Exception ex)
{
  this.Page.Response.Write(ex.Message);
}
```

6. In the Solution Explorer window, right-click **Lab12**, click **Add**, and then click **Add New Item**.
7. Click **Empty Element**.
8. In the **Name** text box, type **ProfileReport**, and then click **Add**.
9. Between the start and end tags of the **<Elements>** element, add the following markup:

```
<CustomAction Id="ProfileActionsMenu"
  GroupId="SiteActions"
  Location="Microsoft.SharePoint.StandardMenu"
  Sequence="1973"
  Title="Profile Reports">
    <UrlAction Url="/_layouts/Lab12/ProfileReporter.aspx" />
</CustomAction>
<CustomAction Id="TaxonomyActionsMenu"
  GroupId="SiteActions"
  Location="Microsoft.SharePoint.StandardMenu"
  Sequence="1981"
  Title="Taxonomy Reports">
    <UrlAction Url="/_layouts/Lab12/TaxonomyReporter.aspx" />
</CustomAction>
```

10. In the Solution Explorer window, right-click **Lab12**, and then click **Deploy**.

▶ Task 4: Test the solution

1. Start Internet Explorer, and browse to **http://sharepoint**.
2. In the top right section of the page, click the **SHAREPOINT\administrator** drop-down menu, and then click **Sign in as Different User**.
3. In the **User Name** text box, type **SHAREPOINT\KrishnaS**.
4. In the **Password** text box, type **P@ssw0rd**, and then click **OK**.
5. On the **Site Actions** menu, click **Profile Reports**.

   Note that the Email and Personal Site columns have values for Krishna.
6. In the top right section of the page, click the **SHAREPOINT\krishnas** drop-down menu, and then click **Sign in as Different User**.
7. In the **User Name** text box, type **SHAREPOINT\MartinR**.
8. In the **Password** text box, type **P@ssw0rd**, and then click **OK**.

   Note that the Email column has a value for Martin, but that he has not yet created his personal site.

9. Click **Create**.
10. On the **Site Actions** menu, click **Profile Reports** to refresh the page.

    Note that the Email and Personal Site columns have values for Martin.
11. In the top right section of the page, click the **SHAREPOINT\martinr** drop-down menu, and then click **Sign in as Different User**.
12. In the **User Name** text box, type **SHAREPOINT\LauraG**.
13. In the **Password** text box, type **P@ssw0rd**, and then click **OK**.

    Note that Laura has not yet set her e-mail address or created her personal site.
14. Click **Set to default**.
15. In the top right section of the page, click the **SHAREPOINT\laurag** drop-down menu, and then click **My Profile**.
16. Below the picture, click **Edit My Profile**.

    Note that Laura's Work Email property has now been set.
17. Browse to **http://sharepoint**.
18. On the **Site Actions** menu, click **Profile Reports**.
19. Click **Create**.
20. On the **Site Actions** menu, click **Profile Reports** to refresh the page.

    Note that the Email and Personal Site columns now both have values for Laura.
21. Close Internet Explorer.

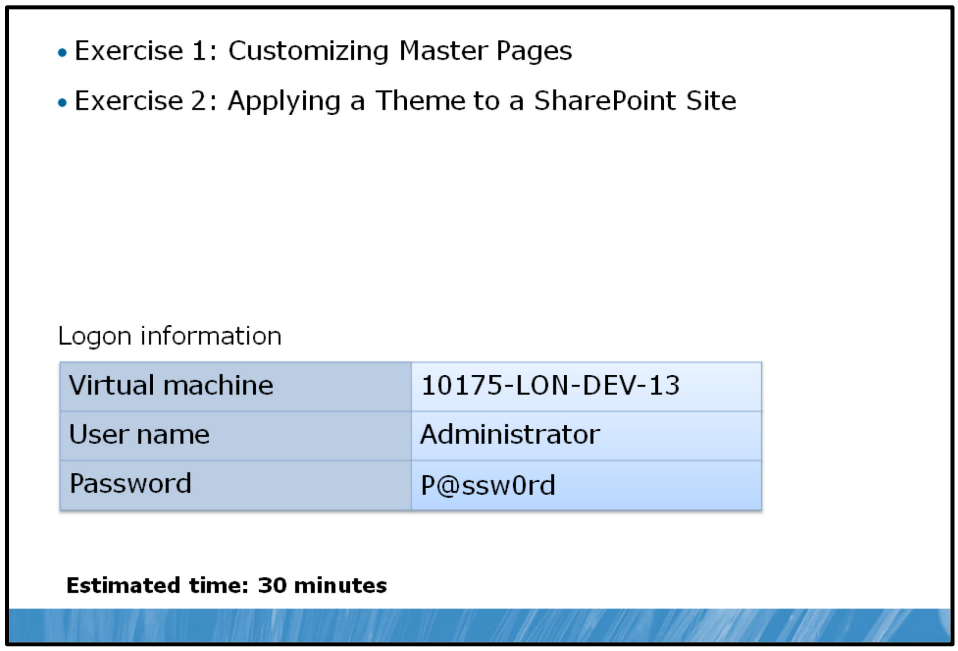
Results: After this exercise, you should have created user profile properties and personal sites programmatically.

Module 13

Lab Instructions: Developing Content Management Solutions

Contents:

Lab: Branding SharePoint Sites



Log On to the Virtual Machine for This Lab

For this lab, you use the available virtual machine environment. Before you begin the lab, you must log on to the **10175-LON-DEV-13** virtual machine as **Administrator** using the password **P@ssw0rd**.

Exercise 1: Customizing Master Pages

▶ Task 1: Edit a SharePoint site by using SharePoint Designer 2010

1. Start Windows® Internet Explorer®, and browse to **http://sharepoint**.
2. On the **Site Actions** menu, click **Edit Site in SharePoint Designer**.
3. In the Site Objects pane, click **Master Pages**.
4. Right-click **v4.master** and then click **Copy**.
5. Press CTRL+V to paste a copy of the **v4.master** file.
6. Right-click **v4_copy(1).master** and then click **Rename**.
7. Rename the file to **NewMaster.master**.
8. Click **NewMaster.master**.

▶ Task 2: Customize and apply a master page to a SharePoint site

1. In the **Customization** section, click **Edit file**.

   The NewMaster.master file is opened in SharePoint Designer.
2. Click the **Design** tab.
3. Near the bottom of the **Quick Launch** bar, click **All Site Content** and then press DELETE.
4. In the **Quick Launch** bar, click **Libraries** and then on the toolbar, click **Align Text Right**.
5. Click **Save** and then click **Yes**.
6. In the Site Objects pane, click **Master Pages**.
7. Right-click **NewMaster.master** and then click **Set as Default Master Page**.
8. Close SharePoint Designer.
9. Switch to Internet Explorer and refresh the page.

   Your customizations have been applied

Exercise 2: Applying a Theme to a SharePoint Site

Task 1: Create a theme by using Microsoft PowerPoint 2010

1. On the **Start** menu, click **All Programs**, click **Microsoft Office** and then click **Microsoft PowerPoint 2010**.
2. On the ribbon, click the **Design** tab.
3. Click the **More** drop-down arrow in the **Themes** section, and then click the theme at the end of the third row. (Hint: The tooltip text for the theme is **Metro**.)
4. On the **File** menu, click **Save As**.
5. In the **Save as type** list, click **Office Theme**.
6. In the left-hand pane, click **Desktop**.
7. In the **File name** text box, type **Metro**.
8. Click **Save**.
9. Close PowerPoint and do not save changes if prompted.

Task 2: Apply the Metro custom theme to the SharePoint site

1. Switch to Internet Explorer and browse to **http://sharepoint**.
2. On the **Site Actions** menu, click **Site Settings**.
3. In the **Galleries** section, click **Themes**.
4. Near the bottom of the page, click **Add New Item**.
5. Click **Browse**.
6. Browse to the **Desktop** folder, and then click **Metro**.
7. Click **Open** and then click **OK**.
8. Click **Save**.
9. On the **Site Actions** menu, click **Site Settings**.
10. In the **Look and Feel** section, click **Site theme**.
11. Click **Metro** and then click **Apply**.
12. Close all applications. You have now completed this lab.

Module 1

Lab Answer Key: Introduction to the SharePoint 2010 Development Platform

Contents:

Lab: Developing with the SharePoint 2010 Object Hierarchy

Log on to the Virtual Machine for this Lab

For this lab, you use the available virtual machine environment. Before you begin the lab, log on to the **10175-LON-DEV-01** virtual machine as **Administrator**, with a password of **P@ssw0rd**.

Exercise 1: Creating SharePoint 2010 Application Pages by Using Visual Studio 2010

▶ Task 1: Create a Visual Studio 2010 project for SharePoint

1. On the **Start** menu, click **All Programs**, click **Microsoft Visual Studio 2010** and then click **Microsoft Visual Studio 2010**.
2. On the **File** menu, point to **New**, and then click **Project**.
3. In the **Installed Templates** section, expand the **Visual C#** node, expand the **SharePoint** node and then click **2010**.
4. Click **Empty SharePoint Project**.
5. In the **Name** text box, type **Lab01**.
6. In the **Location** text box, type **E:\Student\Lab01\Starter**.
7. Click **OK**.

   The SharePoint Customization Wizard appears.
8. In the **What local site do you want to use for debugging?** text box, type **http://sharepoint**.
9. In the **What is the trust level for this SharePoint** solution section, click the **Deploy as a farm solution** option.
10. Click **Finish**.

    The project is created.

▶ Task 2: Add Application pages to the SharePoint 2010 project

1. In the Solution Explorer window, right-click **Lab01**, point to **Add** and then click **New Item**.
2. Click **Application Page**.
3. In the **Name** text box, type **FarmHierarchy.aspx** and then click **Add**.
4. In the Solution Explorer window, right-click **Lab01**, point to **Add** and then click **New Item**.
5. Click **Application Page**.
6. In the **Name** text box, type **PropertyChanger.aspx** and then click **Add**.

▶ Task 3: Add user interface components to SharePoint Application pages

1. Click the **FarmHierarchy.aspx** tab.
2. Add the following markup between the opening and closing tags of the **<asp:Content>** element that has an **ID** of **Main**:

```
<h2>My Farm</h2>
<asp:TreeView ID="farmHierarchyViewer" runat="server"
```

```
  ShowLines="true" EnableViewState="true">
</asp:TreeView>
```

3. Change the markup between the opening and closing tags of the **<asp:Content>** element that has an **ID** of **PageTitle** to the following:

```
Farm Hierarchy and Properties
```

4. Change the markup between the opening and closing tags of the **<asp:Content>** element that has an **ID** of **PageTitleInTitleArea** to:

```
My Farm Hierarchy
```

5. Click the **PropertyChanger.aspx** tab.

6. Add the following markup between the opening and closing tags of the **<asp:Content>** element that has an **ID** of **Main**:

```
<h2>Properties:</h2>
<asp:Label ID="objectName" runat="server" Text="">
</asp:Label><br/><br/>
<asp:Panel ID="webProperties" runat="server" Visible="false"
  BorderColor="Orange" BorderStyle="Dashed" BorderWidth="1">
  <asp:Label ID="WebLabel" runat="server" Text="Web Title">
  </asp:Label><br/>
  <asp:TextBox ID="webTitle" runat="server" EnableViewState="true">
  </asp:TextBox> 
  <asp:Button ID="webTitleUpdate" runat="server" Text="Update"/>
   
  <asp:Button ID="webCancel" runat="server" Text="Cancel" />
</asp:Panel>
<asp:Panel ID="listProperties" runat="server" Visible="false"
  BorderColor="Orange" BorderStyle="Dashed" BorderWidth="1">
  <asp:Label ID="ListLabel" runat="server" Text="List Properties">
  </asp:Label><br/>
  <asp:CheckBox ID="listVersioning" runat="server"
    EnableViewState="true" Text="Enable Versioning" />
  <br/>
  <asp:CheckBox ID="listContentTypes" runat="server"
    EnableViewState="true" Text="Enable Content Types" />
   
  <asp:Button ID="listPropertiesUpdate" runat="server"
    Text="Update" />
   
  <asp:Button ID="listCancel" runat="server" Text="Cancel"/>
</asp:Panel>
```

7. Change the markup between the opening and closing tags of the **<asp:Content>** element that has an **ID** of **PageTitle** so that it reads:

```
Property Changer
```

8. Change the markup between the opening and closing tags of the **<asp:Content>** element that has an **ID** of **PageTitleInTitleArea** so that it reads:

```
My Property Editor
```

9. On the **File** menu, click **Save All**.

Exercise 2: Enumerating SharePoint 2010 Farm Hierarchies

▶ Task 1: Retrieve details of services and Web applications from the SharePoint farm

1. In the Solution Explorer window, right-click **FarmHierarchy.aspx**, and click **View Code**.
2. Near the top of the file, directly beneath the existing **using** statements, add the following code:

```
using System.Web.UI.WebControls;
using Microsoft.SharePoint.Administration;
```

3. Between the opening and closing braces of the **Page_Load** function, add the following code:

```
SPFarm thisFarm = SPFarm.Local;
TreeNode node;
farmHierarchyViewer.Nodes.Clear();
foreach (SPService svc in thisFarm.Services)
{
  node = new TreeNode();
  node.Text = "Farm Service (Type=" + svc.TypeName + "; Status="
    + svc.Status + ")";
  farmHierarchyViewer.Nodes.Add(node);
  TreeNode svcNode = node;
  if (svc is SPWebService)
  {
    SPWebService webSvc = (SPWebService)svc;
    foreach (SPWebApplication webApp in webSvc.WebApplications)
    {
      node = new TreeNode();
      node.Text = webApp.DisplayName;
      svcNode.ChildNodes.Add(node);
      TreeNode webAppNode = node;
      if (!webApp.IsAdministrationWebApplication)
      {
        foreach (SPSite site in webApp.Sites)
        {
          site.CatchAccessDeniedException = false;
          try
          {
            node = new TreeNode();
            node.Text = site.Url;
            webAppNode.ChildNodes.Add(node);
            TreeNode siteNode = node;
            node = new TreeNode(site.RootWeb.Title, null, null,
              site.RootWeb.Url +
"/_layouts/lab01/PropertyChanger.aspx?type=web&objectID="
              + site.RootWeb.ID, "_self");
            siteNode.ChildNodes.Add(node);
            TreeNode parentNode = node;
            foreach (SPList list in site.RootWeb.Lists)
            {
              node = new TreeNode(list.Title, null, null,
                site.RootWeb.Url +
"/_layouts/lab01/PropertyChanger.aspx?type=list&objectID="
                + list.ID, "_self");
              parentNode.ChildNodes.Add(node);
            }
            foreach (SPWeb childWeb in site.RootWeb.Webs)
            {
              try
              {
                addWebs(childWeb, parentNode);
              }
              finally
```

```
              {
                childWeb.Dispose();
              }
            }
            site.CatchAccessDeniedException = false;
          }
          finally
          {
            site.Dispose();
          }
        }
      }
    }
  }
}
farmHierarchyViewer.ExpandAll();
```

4. Below the closing brace for the **Page_Load** function, add the following code:

```
void addWebs(SPWeb web, TreeNode parentNode)
{
  TreeNode node;
  node = new TreeNode(web.Title, null, null, web.Url
    + "/_layouts/lab01/PropertyChanger.aspx?type=web&objectID="
    + web.ID, "_self");
  parentNode.ChildNodes.Add(node);
  parentNode = node;
  foreach (SPList list in web.Lists)
  {
    node = new TreeNode(list.Title, null, null, web.Url
      + "/_layouts/lab01/PropertyChanger.aspx?type=list&objectID="
      + list.ID, "_self");
    parentNode.ChildNodes.Add(node);
  }
  foreach (SPWeb childWeb in web.Webs)
  {
    try
    {
      addWebs(childWeb, parentNode);
    }
    finally
    {
      childWeb.Dispose();
    }
  }
}
```

5. On the **File** menu, click **Save All**.

Exercise 3: Manipulating Properties of Objects in the SharePoint Farm

▶ Task 1: Retrieve properties of SharePoint objects

1. In the Solution Explorer window, right-click **PropertyChanger.aspx**, and click **View Code**.
2. Directly above the **Page_Load** function, add the following code:

```
SPWeb thisWeb = null;
SPList thisList = null;
```

3. Between the opening and closing braces of the **Page_Load** function, add the following code:

```
webTitleUpdate.Click += new EventHandler(webTitleUpdate_Click);
listPropertiesUpdate.Click +=
    new EventHandler(listPropertiesUpdate_Click);
webCancel.Click += new EventHandler(allCancel_Click);
listCancel.Click += new EventHandler(allCancel_Click);
try
{
  string objectType = string.Empty;
  string objectID = string.Empty;
  string objectUrl = string.Empty;
  if (this.Page.Request["type"] != null)
  {
    objectType = this.Page.Request["type"].ToString();
  }
  else
  {
    objectName.Text = "Malformed URL";
    listProperties.Visible = false;
    webProperties.Visible = false;
    return;
  }
  if (this.Page.Request["objectID"] != null)
  {
    objectID = this.Page.Request["objectID"].ToString();
  }
  else
  {
    objectName.Text = "Malformed URL";
    listProperties.Visible = false;
    webProperties.Visible = false;
    return;
  }
  if (objectType == "web")
  {
    listProperties.Visible = false;
    webProperties.Visible = true;
    SPSite thisSite = SPContext.Current.Site;
    thisWeb = thisSite.OpenWeb(new Guid(objectID));
    objectName.Text = "Web: " + thisWeb.Title;
    if (!Page.IsPostBack)
    {
      webTitle.Text = thisWeb.Title;
      thisWeb.Dispose();
      //Do NOT dispose of thisSite!
    }
  }
  if (objectType == "list")
  {
    webProperties.Visible = false;
    listProperties.Visible = true;
    thisWeb = SPContext.Current.Web;
```

```
    thisList = thisWeb.Lists[new Guid(objectID)];
    objectName.Text = "List: " + thisList.Title;
    if (!Page.IsPostBack)
    {
      listVersioning.Checked = thisList.EnableVersioning;
      listContentTypes.Checked = thisList.ContentTypesEnabled;
      //Do NOT dispose of thisWeb!
    }
  }
}
catch (Exception ex)
{
  objectName.Text = ex.Message;
}
```

4. On the **File** menu, click **Save All**.

Task 2: Set properties of SharePoint objects

1. Below the closing brace for the **Page_Load** function, add the following code:

```
void webTitleUpdate_Click(object sender, EventArgs e)
{
  try
  {
    thisWeb.AllowUnsafeUpdates = true;
    thisWeb.Title = webTitle.Text;
    thisWeb.Update();
    thisWeb.AllowUnsafeUpdates = false;
    this.Page.Response.Redirect(thisWeb.Url
      + "/_layouts/lab01/FarmHierarchy.aspx");
  }
  catch(Exception ex)
  {
    objectName.Text = ex.Message;
    listProperties.Visible = false;
    webProperties.Visible = false;
  }
}
```

2. Below the closing brace for the **webTitleUpdate_Click** function, add the following code:

```
void listPropertiesUpdate_Click(object sender, EventArgs e)
{
  try
  {
    thisWeb.AllowUnsafeUpdates = true;
    thisList.EnableVersioning = listVersioning.Checked;
    thisList.ContentTypesEnabled = listContentTypes.Checked;
    thisList.Update();
    thisWeb.AllowUnsafeUpdates = false;
    this.Page.Response.Redirect(thisWeb.Url
      + "/_layouts/lab01/FarmHierarchy.aspx");
  }
  catch (Exception ex)
  {
     objectName.Text = ex.Message;
     listProperties.Visible = false;
     webProperties.Visible = false;
  }
}
```

3. Below the closing brace for the **listPropertiesUpdate_Click** function, add the following code:

```
void allCancel_Click(object sender, EventArgs e)
{
  this.Page.Response.Redirect(thisWeb.Url
    + "/_layouts/lab01/FarmHierarchy.aspx");
}
```

4. On the **File** menu, click **Save All**.

Task 3: Test the solution

1. In the Solution Explorer window, right-click **Lab01** and then click **Deploy**.
2. On the **Start** menu, click **Internet Explorer**.
3. In the **Address Bar**, type **http://sharepoint/_layouts/lab01/farmhierarchy.aspx** and press ENTER.
4. When the page has loaded, review the details shown in the tree-view control.
5. In the tree-view control, click the **Home** node.

   The PropertyChanger.aspx page appears.
6. In the **Web Title** text box, type **HR Intranet** and then click **Update**.

   The title of the Web in the breadcrumb control and in the tree-view control is updated to reflect your changes.
7. In the tree-view control, click the **Shared Documents** node.
8. Check the **Enable Versioning** check box.
9. Check the **Enable Content Types** check box.
10. Click **Update**.
11. In the tree-view control, click the **Tasks** node.
12. Clear the **Enable Content Types** check box.
13. Click **Update**.
14. In the tree-view control, click the **Tasks** node once more.

    The Tasks list currently does not support content types, because your code set this property to false when you clicked Update in step 13.
15. Check the **Enable Content Types** check box.
16. Click **Update**.
17. Close all applications. You have now completed this lab.

Module 2

Lab Answer Key: Using SharePoint 2010 Developer Tools

Contents:

Lab: Using SharePoint 2010 Developer Tools

Log On to the Virtual Machine for This Lab

For this lab, you use the available virtual machine environment. Before you begin the lab, you must log on to the **10175-LON-DEV-02** virtual machine as **Administrator** using the password **P@ssw0rd**.

Exercise 1: Creating Document Libraries by Using SharePoint Designer 2010

▶ Task 1: Edit the HR Intranet site in SharePoint Designer 2010

1. On the **Start** menu, click **Internet Explorer**.

   The Home page for the HR Intranet site appears in Windows® Internet Explorer®.

2. On the **Site Actions** menu, click **Edit in SharePoint Designer**.

   Microsoft® SharePoint® Designer appears, and the site is opened.

3. On the ribbon, click **Document Library**, and then click **Document Library**.

   The Create list or document library dialog box appears.

4. In the **Name** text box, type **Resumes**.
5. In the **Description** text box, type **Library for storing resumes from job applicants**, and then click **OK**.

▶ Task 2: Add a column to the Resumes library

1. In the Site Objects pane, click **Lists and Libraries**.
2. In the **Document Libraries** section, click **Resumes**.
3. In the **Customization** section, click **Edit list columns**.
4. On the ribbon, click **Add New Column**, and then click **Yes/No (checkbox)**.
5. Type **Active** and press ENTER.
6. On the **Quick Access** toolbar, click **Save**.

▶ Task 3: Create a view for the Resumes library

1. In the breadcrumb control, click **Resumes**.
2. In the **Views** section, click **New**.

   The Create New List View dialog box appears.

3. In the **Name** text box, type **Active**, and then click **OK**.
4. In the **Views** section, click **Active** and wait for the page to load.
5. Click the **Design** tab and then click **Type**.

   The PlaceholderMain (Custom) control is highlighted.

6. On the ribbon, click **Filter**.

   The Filter Criteria dialog box appears.

7. In the **Field Name** drop-down list, click **Active**.
8. In the **Comparison** drop-down list, click **Equals**.
9. In the **Value** drop-down list, click **Yes**.

10. Click **OK**.
11. On the **Quick Access** toolbar, click **Save**.
12. Close SharePoint Designer.

▶ Task 4: Test the Resumes library

1. Switch to Internet Explorer, and in the breadcrumb control click **HR Intranet**.
2. In the **Quick Launch** bar, click **Resumes**.

   Note that the Resumes library has a column named Active.
3. In the breadcrumb control, click **All Documents** and note that the drop-down list includes a view called **Active**.
4. Close Internet Explorer.

Exercise 2: Creating SharePoint List Definitions and Instances by Using Visual Studio 2010

Task 1: Create a Visual Studio 2010 project for SharePoint

1. On the **Start** menu, click **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.
2. On the **File** menu, point to **New**, and then click **Project**.
3. In the **Installed Templates** section, expand the **Visual C#** node, expand the **SharePoint** node, and then click **2010**.
4. Click **Empty SharePoint Project**.
5. In the **Name** text box, type **Lab02**.
6. In the **Location** text box, type **E:\Student\Lab02\Starter**.
7. Click **OK**.

   The SharePoint Customization Wizard appears.
8. In the **What local site do you want to use for debugging?** text box, type **http://sharepoint**.
9. In the **What is the trust level for this SharePoint solution** section, click the **Deploy as a farm solution** option.
10. Click **Finish**.

    The project is created.

Task 2: Add a list definition and instance based on the tasks template to the SharePoint 2010 project

1. In the Solution Explorer window, right-click **Lab02**, point to **Add**, and then click **New Item**.
2. Click **List Definition**.
3. In the **Name** text box, type **RecruitmentTasks**, and then click **Add**.

   The SharePoint Customization Wizard appears.
4. In the **What is the display name of the list definition?** text box, type **RecruitmentTasks**.
5. In the **What is the type of the list definition?** drop-down list, click **Tasks**.
6. Ensure that the **Add a list instance for this list definition** check box is selected, and then click **Finish**.

   The Elements.xml file for the RecruitmentTasks list definition is opened.
7. Replace the entire **ListTemplate** element with the following markup:

```
<ListTemplate
  Name="RecruitmentTasks"
  Type="10000"
  BaseType="0"
  OnQuickLaunch="FALSE"
  SecurityBits="11"
  Sequence="360"
  DisplayName="RecruitmentTasks"
  Description="Recruitment Tasks"
  Image="/_layouts/images/ittask.gif"
  AllowDeletion="TRUE"
  AllowEveryoneViewItems="TRUE"
```

```
  DisableAttachments="TRUE"
  NoCrawl="TRUE"
  />
```

8. In the Solution Explorer window, right-click **ListInstance1**, and then click **Rename**.
9. Type **Interviews** and press ENTER.
10. In the Solution Explorer window, expand **Interviews**.
11. Double-click the **Elements.xml** file that is a subnode of **Interviews**.
12. Replace the entire **ListInstance** element with the following markup:

```
<ListInstance Title="Interviews"
  OnQuickLaunch="TRUE"
  TemplateType="10000"
  Url="Lists/Interviews"
  Description="Interviews for External Candidates">
</ListInstance>
```

13. On the **File** menu, click **Save All**.
14. Close all open documents in Microsoft Visual Studio®, but leave the project open and leave Visual Studio running.

▶ Task 3: Add a list definition and instance based on the custom list template to the SharePoint 2010 project

1. In the Solution Explorer window, right-click **Lab02**, point to **Add**, and then click **New Item**.
2. Click **List Definition**.
3. In the **Name** text box, type **CandidateList**, and then click **Add**.

   The SharePoint Customization Wizard appears.
4. In the **What is the display name of the list definition?** text box, type **CandidateList**.
5. In the **What is the type of the list definition?** drop-down list, click **Custom List**.
6. Ensure that the **Add a list instance for this list definition** check box is selected, and then click **Finish**.

   The Elements.xml file for the CandidateList list definition is opened.
7. Replace the entire **ListTemplate** element with the following markup:

```
<ListTemplate
  Name="CandidateList"
  Type="10001"
  BaseType="0"
  OnQuickLaunch="FALSE"
  SecurityBits="11"
  Sequence="410"
  DisplayName="CandidateList"
  Description="External Candidates"
  Image="/_layouts/images/itgen.gif"
  AllowDeletion="TRUE"/>
```

8. In the Solution Explorer window, right-click **ListInstance1**, and then click **Rename**.
9. Type **Candidates** and press ENTER.

10. In the Solution Explorer window, expand **Candidates**.
11. Double-click the **Elements.xml** file that is a subnode of **Candidates**.
12. Replace the entire **ListInstance** element with the following markup:

```
<ListInstance Title="Candidates"
  OnQuickLaunch="TRUE"
  TemplateType="10001"
  Url="Lists/Candidates"
  Description="External Candidates">

</ListInstance>
```

13. On the **File** menu, click **Save All**.
14. Close all open documents in Visual Studio, but leave the project open and leave Visual Studio running.

Task 4: Deploy and manage the solution from Visual Studio 2010

1. In the Solution Explorer window, right-click **Lab02**, and then click **Deploy**.
2. On the **Start** menu, click **Internet Explorer**. Note that the Quick Launch bar shows links to **Interviews** and **Candidates**; these are the list instances you have just deployed.
3. In the **Quick Launch** bar, click **Candidates**, and note that the list does not contain any data.
4. On the ribbon, click the **List** tab.

   If the Internet Explorer dialog box appears and prompts you that content from the website is being blocked, clear the **Continue to prompt when website content is blocked** checkbox, and then click **Close**.
5. On the ribbon, click **List Settings**. In the **Permissions and Management** section, note that there is a link named **Delete this list**.
6. In the breadcrumb control, click **HR Intranet**.
7. Leave Internet Explorer running and switch to Visual Studio.
8. In the Solution Explorer window, double-click **CandidateList** to open its **Elements.xml** file.
9. Edit the **AllowDeletion** attribute so that it reads:

```
AllowDeletion="FALSE"
```

10. In the Solution Explorer window, expand **CandidateList**, and then double-click **Candidates** to open its **Elements.xml** file.
11. Add the following markup *before* the closing tag of the **ListInstance** element:

```
<Data>
  <Rows>
    <Row>
      <Field Name="Title">Geert Camelbeke</Field>
    </Row>
    <Row>
      <Field Name="Title">Ivo Hamels</Field>
    </Row>
    <Row>
      <Field Name="Title">Eduardo Melo</Field>
    </Row>
    <Row>
```

```
      <Field Name="Title">Svetlana Omelchenko</Field>
    </Row>
    <Row>
      <Field Name="Title">Melanie Speckman</Field>
    </Row>
    <Row>
      <Field Name="Title">Steffen Tschimmel</Field>
    </Row>
  </Rows>
</Data>
```

12. On the **File** menu, click **Save All**.
13. In the Solution Explorer window, right-click **Lab02**, and then click **Deploy**.
14. In the **Deployment Conflicts** dialog box, check the **Do not prompt me again for these items** check box, and then click **Resolve Automatically**.
15. Switch to Internet Explorer. Note that the **Quick Launch** bar shows links to **Interviews** and **Candidates**; these are the list instances you have just redeployed.
16. In the **Quick Launch** bar, click **Candidates**, and note that the list now contains data.
17. On the ribbon, click the **List** tab.
18. On the ribbon, click **List Settings**. In the **Permissions and Management** section, note that there is no longer a link named **Delete this list**.
19. In the breadcrumb control, click **HR Intranet**.
20. Leave Internet Explorer running and switch to Visual Studio.

▶ Task 5: Define custom list schema

1. In the Solution Explorer window, right-click **Lab02**, point to **Add**, and then click **New Item**.
2. Click **List Definition**.
3. In the **Name** text box, type **InterviewOutcomes**, and then click **Add**.

   The SharePoint Customization Wizard appears.
4. In the **What is the display name of the list definition?** text box, type **InterviewOutcomes**.
5. In the **What is the type of the list definition?** drop-down list, click **Custom List**.
6. Ensure that the **Add a list instance for this list definition** check box is selected, and then click **Finish**.

   The Elements.xml file for the InterviewOutcomes list definition is opened.
7. Replace the entire **ListTemplate** element with the following markup:

```
<ListTemplate
  Name="InterviewOutcomes"
  Type="10002"
  BaseType="0"
  OnQuickLaunch="FALSE"
  SecurityBits="11"
  Sequence="410"
  DisplayName="InterviewOutcomes"
  Description="Outcomes of Interviews for External Candidates"
  Image="/_layouts/images/itgen.gif"/>
```

8. In the Solution Explorer window, right-click **ListInstance1**, and then click **Rename**.
9. Type **Outcomes** and press ENTER.

10. In the Solution Explorer window, expand **Outcomes**.
11. Double-click the **Elements.xml** file that is a subnode of **Outcomes**.
12. Replace the entire **ListInstance** element with the following markup:

```
<ListInstance Title="Outcomes"
  OnQuickLaunch="TRUE"
  TemplateType="10002"
  Url="Lists/Outcomes"
  Description="Interview Outcomes">
</ListInstance>
```

13. In the Solution Explorer window, double-click **Schema.xml** that is a subnode of **InterviewOutcomes**.
14. Place the cursor between the opening and closing tags of the **<Fields></Fields>** element on line 10 of the file. Then, press ENTER.
15. In the space you have just created, add the following markup:

```
<Field ID="{c3a92d97-2b77-4a25-9698-3ab54874bc06}"
  Name="Candidate" Type="Lookup" DisplayName="Candidate"
  List="Lists/Candidates" ShowField="Title" Required="TRUE">
</Field>
<Field ID="{c3a92d97-2b77-4a25-9698-3ab54874bc19}"
  Name="Interviewer" Type="User" DisplayName="Interviewer"
  List="UserInfo" Required="TRUE" ShowField="ImnName"
  UserSelectionMode="PeopleAndGroups">
</Field>
<Field ID="{c3a92d97-2b77-4a25-9698-3ab54874bc81}"
  Name="Offer" Type="Boolean" DisplayName="Job Offered?"
  Required="TRUE">
</Field>
```

16. Press CTRL+F.
17. In the **Find what** text box, type **LinkTitleNoMenu**, and then click **Find Next**.

    Visual Studio finds markup that reads: <FieldRef Name=" LinkTitleNoMenu ">
    </FieldRef>

18. Close the **Find and Replace** dialog box.
19. Place the cursor at the end of the line that Visual Studio has found, and then press ENTER.
20. Add the following markup to the space you have just created:

```
<FieldRef Name="Candidate"></FieldRef>
<FieldRef Name="Interviewer"></FieldRef>
<FieldRef Name="Offer"></FieldRef>
```

21. Press CTRL+F.
22. In the **Find what** text box, type **LinkTitle**, and then click **Find Next**.

    Visual Studio finds markup that reads: <FieldRef Name=" LinkTitle ">
    </FieldRef>

23. Close the **Find and Replace** dialog box.
24. Place the cursor at the end of the line that Visual Studio has found, and then press ENTER.
25. Add the following markup to the space you have just created:

```
<FieldRef Name="Candidate"></FieldRef>
<FieldRef Name="Interviewer"></FieldRef>
<FieldRef Name="Offer"></FieldRef>
```

26. On the **File** menu, click **Save All**.

Exercise 3: Packaging Features and Solutions by Using Visual Studio 2010

Task 1: Remove items from a feature

1. In the Solution Explorer window, expand **Features**.
2. Double-click **Feature1**.

   The Feature Designer for Feature1 appears.
3. In the Items in the Feature pane, double-click **InterviewOutcomes**.

   InterviewOutcomes is removed from the Items in the Feature pane.
4. In the Items in the Feature pane, double-click **Outcomes**.

   Outcomes is removed from the Items in the Feature pane.
5. On the **File** menu, click **Save All**.
6. Close **Feature1**.

Task 2: Create a new feature and add items to it

1. In the Solution Explorer window, right-click **Features**, and click **Add Feature**.

   The Feature Designer for Feature2 appears.
2. In the Solution Explorer window, right-click **Feature2**, and click **Rename**.
3. Type **Interviews** and press ENTER.
4. In the Items in the Solution pane, double-click **InterviewOutcomes**.

   InterviewOutcomes is added to the Items in the Feature pane.
5. In the Items in the Solution pane, double-click **Outcomes**.

   Outcomes is added to the Items in the Feature pane.
6. In the **Title** text box, type **Recruitment Interviews**.
7. In the **Description** text box, type **List definition and instance for the interview process**.
8. On the **File** menu, click **Save All**.
9. Close the **Interviews** feature.

Task 3: Package Features

1. In the Solution Explorer window, double-click **Package**.
2. In the Items in the Package pane, double-click **Lab02 Feature1**.

   The feature is removed from the Items in the Package pane.
3. On the **File** menu, click **Save All**.
4. Close the Package Designer.
5. In the Solution Explorer window, right-click **Lab02**, and then click **Package**.

   Visual Studio builds and packages the solution, but does not deploy it.
6. Leave Visual Studio running.
7. Using Windows Explorer, navigate to the following folder:
   E:\Student\Lab02\Starter\Lab02\Lab02\bin\debug

   Visual Studio has created a solution file called Lab02.wsp.
8. Drag **Lab02.wsp** to the **Local Disk (C:)** folder.

Task 4: Deploy solutions by using Windows PowerShell

1. On the **Start** menu, click **All Programs**, click **Microsoft SharePoint 2010 Products**, and then click **SharePoint 2010 Management Shell**.

   The Windows PowerShell® command prompt appears.
2. At the command prompt, type **Add-SPSolution C:\lab02.wsp**, and press ENTER.

   Windows PowerShell displays a message to inform you that a solution named Lab02.wsp already exists. This is the package that you previously deployed from Visual Studio.
3. At the command prompt, type **Remove-SPSolution -Identity Lab02.wsp**, and press ENTER.

   Windows PowerShell displays a message to inform you that the solution has already been deployed and must be retracted before it can be removed.
4. At the command prompt, type **Uninstall-SPSolution -Identity Lab02.wsp**, and press ENTER.
5. Press Y and then press ENTER.
6. At the command prompt, type **Remove-SPSolution -Identity Lab02.wsp**, and press ENTER. Press Y and then press ENTER.
7. At the command prompt, type **Add-SPSolution C:\Lab02.wsp**, and press ENTER.
8. At the command prompt, type **Install-SPSolution -Identity Lab02.wsp –GACDeployment**, and press ENTER.
9. At the command prompt, type **Install-SPSolution -Identity Lab02.wsp -GACDeployment –Force**, and press ENTER.
10. Close Windows PowerShell.

Task 5: Finalize the solution

1. Switch to Visual Studio.
2. In the Solution Explorer window, double-click **Package**.
3. In the Items in the Solution pane, double-click **Lab02 Feature1**.

The feature is added to the Items in the Package pane.

4. On the **File** menu, click **Save All**.
5. Close the Package Designer.
6. In the Solution Explorer window, right-click **Lab02**, and then click **Deploy**.

   A deployment error occurs because of the deployment you have just performed by using PowerShell, but you simply need to redeploy one more time.
7. In the Solution Explorer window, right-click **Lab02**, and then click **Deploy**.
8. In the **Deployment Conflicts** dialog box, check the **Do not prompt me again for these items** check box, and then click **Resolve Automatically**.

   Visual Studio builds, packages, and deploys the solution.
9. Switch to Internet Explorer.
10. In the breadcrumb control, click **HR Intranet**.
11. On the **Site Actions** menu, click **Edit in SharePoint Designer**.

    SharePoint Designer appears, and the site is opened.
12. In the Site Objects pane, click **Lists and Libraries**.
13. In the **Document Libraries** section, click **Resumes**.
14. In the **Customization** section, click **Edit list columns**.
15. On the ribbon, click **Add New Column**, and then click **Lookup (Information Already on This Site)**.
16. In the **List or document library** drop-down list, click **Candidates**.
17. In the **Field** drop-down list, click **ID**.
18. In the **Add a column to show each of these additional fields** list, select the **Title** check box, and then click **OK**.
19. Right-click **NewColumn1**, and click **Rename**.
20. Type **CandidateID**, and press ENTER.
21. On the **Quick Access** toolbar, click **Save**.
22. Close SharePoint Designer.

▶ Task 6: Test the solution

1. Switch to Internet Explorer.
2. In the **Quick Launch** bar, click **Resumes** and note the lookup columns that have been added to the library.
3. In the **Quick Launch** bar, click **Candidates** and note the list of candidates.
4. In the **Quick Launch** bar, click **Outcomes**.
5. On the ribbon, click the **List** tab.
6. On the ribbon, click **List Settings**.
7. In the **Views** section, click **All Items**.
8. Expand the **Inline Editing** section, and select the **Allow inline editing** check box.
9. Click **OK**.

10. Below the list columns, click the plus sign (**+**).
11. In the **Title** text box, type **Senior Developer**.
12. In the **Candidate** drop-down list, click **Melanie Speckman**.

    The list of names is retrieved by the lookup field you defined in Visual Studio.
13. In the **Interviewer** text box, type **KrishnaS**, and then click **Check Names**.

    The people-picker functionality is provided by the User field you defined in Visual Studio.
14. Select the **Job Offered?** check box.
15. Click **Save** for the list item.
16. Close all applications.

    You have now completed this lab.

Module 3

Lab Answer Key: Developing SharePoint 2010 Web Parts

Contents:

Lab: Creating SharePoint 2010 Web Parts by Using Visual Studio 2010

Log On to the Virtual Machine for This Lab

For this lab, use the available virtual machine environment. Before you begin the lab, log on to the **10175-LON-DEV-03** virtual machine as **Administrator**, with a password of **P@ssw0rd**.

Exercise 1: Creating, Deploying, and Debugging a Simple Web Part by Using Visual Studio 2010

▶ Task 1: Create an empty SharePoint project

1. On the **Start** menu, click **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.
2. On the **File** menu, point to **New**, and then click **Project**.
3. In the **Installed Templates** section, expand the **Visual C#** node, expand the **SharePoint** node, and then click **2010**.
4. Click **Empty SharePoint Project**.
5. In the **Name** text box, type **Lab03**.
6. In the **Location** text box, type **E:\Student\Lab03\Starter**.
7. Click **OK**.

   The SharePoint Customization Wizard appears.
8. In the **What local site do you want to use for debugging?** text box, type **http://sharepoint**.
9. In the **What is the trust level for this SharePoint** solution section, click the **Deploy as a farm solution** option.
10. Click **Finish**.

    The project is created.

▶ Task 2: Add a standard Web Part to the project

1. In the Solution Explorer window, right-click **Lab03**, point to **Add**, and then click **New Item**.
2. Click **Web Part**.

> **Note:** Do not click Visual Web Part.

3. In the **Name** text box, type **TaskView** and then click **Add**.

   A new Web Part is added to the project, and the class file for the Web Part appears.
4. Place the cursor between the opening and closing braces of the **CreateChildControls** method, and then press ENTER.
5. In the space you have created, type the following code:

```
LiteralControl myMessage =
  new LiteralControl("<H5>People and Tasks</H5>");
this.Controls.Add(myMessage);
```

6. Click in the first line of code that you have just typed.

7. On the **Debug** menu, click **Toggle Breakpoint**.
8. On the **File** menu, click **Save All**.

Task 3: Debug the Web Part

1. In the Solution Explorer window, right-click **Lab03** and then click **Deploy**.

   Microsoft SharePoint builds, packages, and deploys the Web Part.
2. Start Windows® Internet Explorer®.
3. On the **Site Actions** menu, click **New Page**.
4. In the **New page name** text box, type **Recruitment** and then click **Create**.
5. On the ribbon, click the **Insert** tab.
6. On the ribbon, click **Web Part**.
7. In the **Categories** section, click **Custom**.
8. In the **Web Parts** section, click **TaskView**.
9. Click **Add**.

   The Web Part is added to the page.
10. On the ribbon, click **Save and Close**.
11. Close Internet Explorer.
12. Switch to Microsoft Visual Studio®.
13. Press F5.

    SharePoint builds, packages, and deploys the Web Part, and then attaches the debugger to the w3p3.exe process.
14. If the **Debugging Not Enabled** dialog box appears, click **Modify the Web.config file to enable debugging** option, and then click **OK**.

    Internet Explorer starts.
15. On the **Quick Launch** bar, click **Site Pages**.
16. Click **Recruitment**.

    Visual Studio becomes active, with code execution paused on the breakpoint you added in the previous task.
17. In Visual Studio, on the **Debug** toolbar, click **Step Into**.

    If a message box appears informing you that the web server process that was being debugged has been terminated by Internet Information Services, click **OK**; you will then need to stop the debugger, close Internet Explorer, and start from step 13 again. If no message box appears, continue with step 18.
18. On the **Debug** toolbar, click **Continue**.

    Internet Explorer displays the home page with your Web Part.
19. Close Internet Explorer.

    The debugger in Visual Studio detaches from the w3wp.exe process and execution of the Web Part project ends.

Exercise 2: Using SharePoint Components in a Web Part

Task 1: Create class-level variables for the Web Part

1. Place the cursor at the beginning of the line of that reads: **protected override void CreateChildControls** and then press ENTER.
2. In the space you have just created, add the following code:

```
DateTimeControl filterDate;
ListViewByQuery MyCustomView;
SPQuery query;
```

3. On the line *above* the namespace declaration, add the following statement:

```
using Microsoft.SharePoint.Utilities;
```

4. On the **File** menu, click **Save All**.

Task 2: Develop the CreateChildControls method to use SharePoint components

1. Place the cursor at the end of the last line of code that you previously added to the **CreateChildControls** method, and then press ENTER.
2. In the space that you have just created, add the following code:

```
filterDate = new DateTimeControl();
filterDate.DateOnly = true;
filterDate.AutoPostBack = true;
```

3. Press ENTER to create a new line.
4. Type **filterDate.DateChanged +=**.
5. Press TAB.
6. Press TAB.

Visual Studio enters the event handler for you.

7. Place the cursor at the end of the last line of code that you previously added to the **CreateChildControls** method, and then press ENTER.
8. In the space that you have just created, add the following code:

```
SPWeb thisWeb = SPContext.Current.Web;
MyCustomView = new ListViewByQuery();
MyCustomView.List = thisWeb.Lists["Interviews"];
query = new SPQuery(MyCustomView.List.DefaultView);
query.ViewFields = "<FieldRef Name='Title' />"
  + "<FieldRef Name='AssignedTo' /><FieldRef Name='DueDate' />";
MyCustomView.Query = query;
LiteralControl filterMessage
  = new LiteralControl("Tasks due on or before:");
this.Controls.Add(filterMessage);
this.Controls.Add(new LiteralControl("<br />"));
this.Controls.Add(filterDate);
this.Controls.Add(new LiteralControl("<br />"));
this.Controls.Add(MyCustomView);
```

Task 3: Add event handler code for SharePoint components

1. Delete the existing line of code in the **filterDate_DateChanged** event handler.

2. Add the following code to the **filterDate_DateChanged** event handler:

```
string camlQuery = "<Where><Leq><FieldRef Name='DueDate' />"
  + "<Value Type='DateTime'>"
  + SPUtility.CreateISO8601DateTimeFromSystemDateTime
  (filterDate.SelectedDate)
  + "</Value></Leq></Where>";
query.Query = camlQuery;
MyCustomView.Query = query;
```

3. On the **File** menu, click **Save All**.

Task 4: Test the Web Part

1. In the Solution Explorer window, right-click **Lab03** and then click **Deploy**.

   SharePoint builds, packages, and deploys the Web Part.

2. On the **Start** menu, click **Internet Explorer**.
3. On the **Quick Launch** bar, click **Site Pages**.
4. Click **Recruitment**.

   The page shows your updated Web Part.

5. On the ribbon, click **Navigate Up**, and then click **HR Intranet**.
6. On the **Quick Launch** bar, click **Site Pages**.
7. Click **Recruitment**.

   Your Web Part currently shows all tasks in the Interviews list.

8. In your Web Part, click the date picker control.
9. Click the date that represents 14 May, 2011.

   Your Web Part currently shows the task in the Interviews list for interviewing Svetlana.

10. In your Web Part, click the date picker control.
11. Click the date that represents 19 June, 2011.

    Your Web Part currently shows the tasks in the Interviews list for interviewing Svetlana and Ivo.

12. Close Internet Explorer.

Exercise 3: Creating a Visual Web Part by Using Visual Studio 2010

Task 1: Add a Visual Web Part to the project and design its graphical user interface

1. In the Solution Explorer window, right-click **Lab03**, point to **Add**, and then click **New Item**.
2. Click **Visual Web Part**.

> **Note:** Do not click Web Part.

3. In the **Name** text box, type **Overview** and then click **Add**.

   A new Visual Web Part is added to the project, and the OverviewUserControl.ascx markup file for the user control in the Visual Web Part appears.

4. Add the following markup to the end of the **OverviewUserControl.ascx** file:

```
<asp:TreeView ID="overviewTree"
  runat="server"
  ShowLines="true"
  EnableViewState="true">
</asp:TreeView>
```

5. On the **File** menu, click **Save All**.

Task 2: Develop the code for the Visual Web Part

1. On the **View** menu, click **Code**.
2. On the line *above* the namespace declaration, add the following statement:

```
using Microsoft.SharePoint;
```

3. Add the following code between the start and end braces of the **Page_Load** method:

```
try
{
  overviewTree.Nodes.Clear();
  SPWeb thisWeb = SPContext.Current.Web;
  SPList candidates = thisWeb.Lists["Outcomes"];
  SPQuery itemMashup = new SPQuery();
  itemMashup.Joins =
    "<Join Type='LEFT' ListAlias='Candidates'><Eq>"
    + "<FieldRef Name='Candidate' RefType='Id' />"
    + "<FieldRef List='Candidates' Name='ID' /></Eq>"
    + "</Join>";
  itemMashup.ProjectedFields =
    "<Field Name='Applicant' Type='Lookup' List='Candidates'"
      + " ShowField='Title' />"
    + "<Field Name='HomeCity' Type='Lookup' List='Candidates'"
      + " ShowField='HomeCity' />";
  itemMashup.ViewFields = "<FieldRef Name='Applicant' />"
    + "<FieldRef Name='HomeCity' />"
    + "<FieldRef Name='Title' />"
    + "<FieldRef Name='Interviewer' />"
    + "<FieldRef Name='Offer' />";
  SPListItemCollection allCandidates =
    candidates.GetItems(itemMashup);
  foreach (SPListItem item in allCandidates)
  {
   TreeNode applicant = new TreeNode(item["Applicant"].ToString(),
```

```
      null,null,thisWeb.Lists["Candidates"].DefaultViewUrl,
      "_self");
    TreeNode opportunity = new TreeNode(item["Title"].ToString(),
     null, null, thisWeb.Lists["Outcomes"].DefaultViewUrl,
      "_self");
    TreeNode homeCity = new TreeNode(item["HomeCity"].ToString(),
      null, null, thisWeb.Lists["Candidates"].DefaultViewUrl,
      "_self");
    TreeNode interviewer = new TreeNode("Interviewed by: " +
      item["Interviewer"].ToString(), null, null,
      thisWeb.Lists["Interviews"].DefaultViewUrl, "_self");
    TreeNode offered =
      new TreeNode(bool.Parse(item["Offer"].ToString()) == true ?
      "Job Offered" : "Rejected", null, null,
      thisWeb.Lists["Outcomes"].DefaultViewUrl, "_self");
    applicant.ChildNodes.Add(opportunity);
    applicant.ChildNodes.Add(homeCity);
    applicant.ChildNodes.Add(interviewer);
    applicant.ChildNodes.Add(offered);
    overviewTree.Nodes.Add(applicant);
   }
   overviewTree.ExpandAll();
}
catch (Exception ex)
{
   overviewTree.Nodes.Add(new TreeNode("Err" + ex.Message));
}
```

4. On the **File** menu, click **Save All**.

Task 3: Deploy and test the Visual Web Part

1. In the Solution Explorer window, right-click **Lab03** and then click **Deploy**.

   SharePoint builds, packages, and deploys the Web Part.

2. On the **Start** menu, click **Internet Explorer**.
3. On the **Quick Launch** bar, click **Site Pages**.
4. Click **Recruitment**.
5. On the ribbon, click **Edit**.
6. On the ribbon, click the **Insert** tab.
7. On the ribbon, click **Web Part**.
8. In the **Categories** section, click **Custom**.
9. In the **Web Parts** section, click **Overview**.
10. Click **Add**.

    The Web Part is added to the page.

11. On the ribbon, click **Save and Close**.
12. Expand all the nodes in the tree view control to view the data from the lists in HR Intranet site.

Module 4

Lab Answer Key: Working with SharePoint Objects on the Server

Contents:

Lab: Creating and Manipulating Server-Side Objects

Log On to the Virtual Machine for This Lab

For this lab, you use the available virtual machine environment. Before you begin the lab, you must log on to the **10175-LON-DEV-04** virtual machine as **Administrator** using the password **P@ssw0rd**.

Exercise 1: Creating and Securing Sites Programmatically

▶ Task 1: Create an empty SharePoint project

1. On the **Start** menu, click **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.
2. On the **File** menu, point to **New**, and then click **Project**.
3. In the **Installed Templates** section, expand the **Visual C#** node, expand the **SharePoint** node, and then click **2010**.
4. Click **Empty SharePoint Project**.
5. In the **Name** text box, type **Lab04**.
6. In the **Location** text box, type **E:\Student\Lab04\Starter**. Click **OK**.

   The SharePoint Customization Wizard appears.
7. In the **What local site do you want to use for debugging?** text box, type **http://sharepoint**.
8. In the **What is the trust level for this SharePoint solution** section, click the **Deploy as a farm solution** option.
9. Click **Finish**.

   The project is created.

▶ Task 2: Add an application page for creating and securing sites

1. In the Solution Explorer window, right-click **Lab04**, point to **Add**, and then click **New Item**.
2. Click **Application Page**.
3. In the **Name** text box, type **CreateJobDef.aspx**, and then click **Add**.
4. Add the following markup between the opening and closing tags of the **<asp:Content>** element that has an **ID** of **Main**:

```
<asp:Button ID="creator" runat="server"
  Text="Create Job Definition List"/>
<br />
<asp:Label ID="status" runat="server" Text=""></asp:Label>
```

5. Change the markup between the opening and closing tags of the **<asp:Content>** element that has an **ID** of **PageTitle** to the following:

```
Job Definitions
```

6. Change the markup between the opening and closing tags of the **<asp:Content>** element that has an **ID** of **PageTitleInTitleArea** to the following:

```
Utility for creating Job Definitions
```

7. On the **View** menu, click **Code**.
8. Place the cursor between the opening and closing braces of the **Page_Load** function, and then press ENTER.
9. On the new line that you have just created, type **creator.Click +=**.
10. Press TAB.
11. Press TAB.

    Microsoft® Visual Studio® enters the event handler for you.
12. After the closing brace of the **creator_Click** function, add the following function:

```
void createList(SPWeb subWeb)
{

}
```

Note: You develop the **createList** function later in this lab.

13. Replace the existing code in the **creator_Click** function with the following:

```
SPWeb thisWeb = SPContext.Current.Web;
SPWeb newWeb = null;
if (!thisWeb.Webs["JobData"].Exists)
{
  try
  {
    newWeb = thisWeb.Webs.Add("JobData",
      "Job Data",
      "Data for Jobs",
      1033,
      "STS#1",
      true,
      false);
    SPRoleAssignment roleAssign =
      new SPRoleAssignment(@"SHAREPOINT\KrishnaS",
      "krishnas@sharepoint.com", @"SHAREPOINT\KrishnaS",
      "HR Manager");
    SPRoleDefinition roleDef =
      newWeb.RoleDefinitions["Contribute"];
    roleAssign.RoleDefinitionBindings.Add(roleDef);
    newWeb.RoleAssignments.Add(roleAssign);
    newWeb.Update();
    roleAssign = new SPRoleAssignment(@"SHAREPOINT\MartinR",
      "martinr@sharepoint.com", @"SHAREPOINT\MartinR",
      "HR Manager");
    roleDef = newWeb.RoleDefinitions["Contribute"];
    roleAssign.RoleDefinitionBindings.Add(roleDef);
    newWeb.RoleAssignments.Add(roleAssign);
    newWeb.Update();
    createList(newWeb);
    status.Text =
      "Job Data site and Job Definitions list have been added";
    creator.Enabled = false;
  }
  catch (Exception webEx)
  {
    status.Text = webEx.Message;
  }
```

```
  finally
  {
    newWeb.Dispose();
  }
  return;
}
```

14. After the code you have just typed, add the following code:

```
using (SPWeb existingWeb = thisWeb.Webs["JobData"])
{
  try
  {
    SPList jobDefToCreate = existingWeb.Lists.TryGetList
      ("Job Definitions");
    if(jobDefToCreate != null)
    {
      status.Text =
        "Job Data site and Job Definitions already exist";
      creator.Enabled = false;
      return;
    }
    createList(existingWeb);
    status.Text = "Job Data site already exists. "
      + "Job Definitions list have been added";
    creator.Enabled = false;
  }
  catch(Exception listEx)
  {
    status.Text = listEx.Message;
  }
}
```

15. On the **File** menu, click **Save All**.

Exercise 2: Creating Lists Programmatically

▶ Task 1: Complete the code for the **createList** function

1. Add the following code to the **createList** function:

```
Guid jobDefGuid = subWeb.Lists.Add
  ("Job Definitions", "Jobs, Salary Ranges, and Descriptions",
  SPListTemplateType.GenericList);
SPList jobDef = subWeb.Lists[jobDefGuid];
jobDef.Fields.Add("MinSalary", SPFieldType.Currency, true);
jobDef.Fields.Add("MaxSalary", SPFieldType.Currency, true);
jobDef.Fields.Add("JobDescription", SPFieldType.Text, false);
jobDef.OnQuickLaunch = true;
jobDef.Update();

SPView vw = jobDef.Views["All Items"];
vw.ViewFields.Add("MinSalary");
vw.ViewFields.Add("MaxSalary");
vw.ViewFields.Add("JobDescription");
vw.InlineEdit = "TRUE";
vw.Update();

SPListItem newDef;
newDef = jobDef.Items.Add();
newDef["Title"]= "Developer";
newDef["MinSalary"] = "40000";
newDef["MaxSalary"] = "80000";
newDef["JobDescription"] = "SharePoint or other Web Developer";
newDef.Update();

newDef = jobDef.Items.Add();
newDef["Title"] = "Analyst";
newDef["MinSalary"] = "40000";
newDef["MaxSalary"] = "90000";
newDef["JobDescription"] = "Business Analyst";
newDef.Update();

newDef = jobDef.Items.Add();
newDef["Title"] = "Lead Developer";
newDef["MinSalary"] = "60000";
newDef["MaxSalary"] = "100000";
newDef["JobDescription"] = "Developer and Team Leader";
newDef.Update();
newDef = jobDef.Items.Add();
newDef["Title"] = "Solution Architect";
newDef["MinSalary"] = "80000";
newDef["MaxSalary"] = "120000";
newDef["JobDescription"] = "Experienced Solution Architect";
newDef.Update();
```

2. On the **File** menu, click **Save All**.

▶ Task 2: Deploy and test the application page

1. In the Solution Explorer window, right-click **Lab04**, and then click **Deploy**.
2. On the **Start** menu, click **Internet Explorer**.
3. In the **Address Bar**, type **http://sharepoint/_layouts/lab04/CreateJobDef.aspx**, and press ENTER.
4. Click **Create JobDefinition List**.

After a few seconds you will be notified that the site and list have been created.

5. On the **Quick Launch** bar, click **All Site Content**.
6. In the **Sites and Workspaces** section, click **Job Data**.
7. On the **Quick Launch** bar, click **Job Definitions**, and review the data.
8. On the **Site Actions** menu, click the **Site Permissions**.

   Note that the site has unique permissions, and note that KrishnaS and MartinR have been granted contribute permissions.
9. Close Internet Explorer.

Exercise 3: Retrieving Secured Data

Task 1: Create a pair of connected Web Parts

1. Switch to Visual Studio.
2. In the Solution Explorer window, right-click **Lab04**, point to **Add**, and then click **New Item**.
3. Click **Web Part**.
4. In the **Name** text box, type **JobDefinitions**, and then click **Add**.
5. In the Solution Explorer window, right-click **Lab04**, point to **Add**, and then click **New Item**.
6. Click **Web Part**.
7. In the **Name** text box, type **OpenPositions**, and then click **Add**.

Task 2: Create an interface for the connection between the Web Parts

1. Click the **JobDefinitions.cs** tab.
2. Place the cursor immediately after the opening brace of the namespace **Lab04.JobDefinitions**, and then press ENTER.
3. In the space you have just created, type the following code:

```
public interface IJobDef
{
  string JobDef
  {
    get;
    set;
  }
}
```

Task 3: Implement the provider Web Part to retrieve data from the secured subsite

1. Edit the line of code that reads **public class JobDefinitions : WebPart** so that it reads as follows:

```
public class JobDefinitions : WebPart, IJobDef
```

2. Place the cursor immediately before the **CreateChildControls** function declaration, and then press ENTER.
3. In the space you have just created, type the following code:

```
private DropDownList allJobs;
private string _JobDef;
public string JobDef
{
  get
  {
    return (_JobDef);
  }
  set
  {
    _JobDef = value;
  }
}
```

4. After the closing brace of the **JobDef** property, add the following code:

```
[ConnectionProvider("Job")]
```

```
public IJobDef SendJobName()
{
    return (this);
}
```

5. Add the following code to the **CreateChildControls** function:

```
allJobs = new DropDownList();
allJobs.AutoPostBack = true;
allJobs.EnableViewState = true;
populateList();
```

You create the **populateList** function later in this exercise.

6. After the code you have just typed, type **allJobs.SelectedIndexChanged +=**.
7. Press TAB.
8. Press TAB.

Visual Studio enters the event handler for you.

9. Add the following code to the code you have already typed in the **CreateChildControls** function:

```
this.Controls.Add(new LiteralControl
  ("Job Definition Filter:<br/>"));
this.Controls.Add(allJobs);
```

10. Replace the existing code in the **allJobs_SelectedIndexChanged** event handler function with the following:

```
_JobDef = allJobs.SelectedValue;
```

11. Add the following function after the closing brace of the **CreateChildControls** function:

```
void populateList()
{
  SPWeb jobDefWeb=null;
  SPWeb thisWeb = SPContext.Current.Web;
  SPList jobDefList;
  thisWeb.Site.CatchAccessDeniedException = false;
  try
  {
    allJobs.Items.Add("Pick a Job");
    allJobs.Items.Add("All Jobs");
    jobDefWeb = SPContext.Current.Web.Webs["JobData"];
    jobDefList = jobDefWeb.Lists["Job Definitions"];
    foreach (SPListItem item in jobDefList.Items)
    {
      allJobs.Items.Add(item.Title);
    }
  }
  catch (UnauthorizedAccessException secEx)
  {
    SPUser privilegedAccount =
      SPContext.Current.Web.AllUsers[@"SHAREPOINT\SYSTEM"];
    SPUserToken privilegedToken = privilegedAccount.UserToken;
    using (SPSite elevatedSite =
      new SPSite(SPContext.Current.Web.Url, privilegedToken))
    {
      using (SPWeb elevatedWeb = elevatedSite.OpenWeb())
      {
```

```
        jobDefWeb = elevatedWeb.Webs["JobData"];
        jobDefList = jobDefWeb.Lists["Job Definitions"];
        foreach (SPListItem item in jobDefList.Items)
        {
          allJobs.Items.Add(item.Title);
        }
      }
    }
  }
  finally
  {
    thisWeb.Site.CatchAccessDeniedException = true;
    jobDefWeb.Dispose();
  }
}
```

12. On the **File** menu, click **Save All**.

Task 4: Generate LINQ entities for use in the consumer Web Part

1. Click **Start**, and then click **Run**.
2. Type **cmd**, and then press ENTER.
3. At the command prompt, type **cd "C:\Program Files\Common Files \Microsoft Shared\Web Server Extensions\14\bin"**, and press ENTER.
4. At the command prompt, type **SPMetal /web:http://sharepoint /code:E:\Student\HREntities.cs /language:csharp**, and press ENTER.

   You are notified that content types for list form templates were excluded. This is expected behavior.
5. Close the command prompt.

Task 5: Implement the consumer Web Part

1. Switch to Visual Studio.
2. In the Solution Explorer window, right-click **Lab04**, and then click **Add Reference**.
3. Click the **Browse** tab, and then browse to the **C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\14\ISAPI** folder.
4. Click **Microsoft.SharePoint.Linq.dll** and then click **OK**.
5. In the Solution Explorer window, right-click **Lab04**, point to **Add**, and then click **Existing Item**.
6. Browse to the **E: Student** folder.
7. Click **HREntities.cs**, and then click **Add**.
8. Click the **OpenPositions** tab.
9. In the space immediately above the namespace **Lab04.OpenPositions** declaration, add the following **using** statements:

```
using System.Linq;
using Microsoft.SharePoint.Linq;
```

10. Place the cursor immediately before the **CreateChildControls** function declaration, and then press ENTER.
11. In the space you have just created, type the following code:

```
private string jobDefinition = "All Jobs";
```

```
[ConnectionConsumer("Job")]
public void GetJob(Lab04.JobDefinitions.IJobDef Job)
{
  if (Job != null)
  {
    jobDefinition = Job.JobDef;
  }
}
```

12. Add the following code to the **CreateChildControls** function:

```
Table jobTable = new Table();
TableRow jobRow = new TableRow();
TableCell jobDetail = new TableCell();
jobDetail.Text = "Job";
jobDetail.Font.Bold = true;
jobDetail.HorizontalAlign = HorizontalAlign.Center;
jobRow.Cells.Add(jobDetail);

jobDetail = new TableCell();
jobDetail.Text = "Location";
jobDetail.Font.Bold = true;
jobDetail.HorizontalAlign = HorizontalAlign.Center;
jobRow.Cells.Add(jobDetail);
jobDetail = new TableCell();
jobDetail.Text = "Interviewer";
jobDetail.Font.Bold = true;
jobDetail.HorizontalAlign = HorizontalAlign.Center;
jobRow.Cells.Add(jobDetail);

jobTable.Rows.Add(jobRow);

this.Controls.Add(new LiteralControl("Open Positions: " +
  jobDefinition + "<br />"));
HREntitiesDataContext hrDC = new
  HREntitiesDataContext("http://sharepoint");
if (jobDefinition != "All Jobs")
{
  var filteredJobs = from vacancyList in hrDC.CurrentVacancies
    where(vacancyList.Title.Equals(jobDefinition))
    orderby vacancyList.Title
    select new { vacancyList.Title,
      vacancyList.Location, vacancyList.InterviewerImnName };
  foreach (var job in filteredJobs)
  {
    jobRow = new TableRow();
    jobDetail = new TableCell();
    jobDetail.Text = job.Title;
    jobRow.Cells.Add(jobDetail);

    jobDetail = new TableCell();
    jobDetail.Text = job.Location;
    jobRow.Cells.Add(jobDetail);

    jobDetail = new TableCell();
    jobDetail.Text = job.InterviewerImnName;
    jobRow.Cells.Add(jobDetail);

    jobTable.Rows.Add(jobRow);
  }
  this.Controls.Add(jobTable);
  return;
}
```

```
var unfilteredJobs = from vacancyList in hrDC.CurrentVacancies
  where (vacancyList.Title.Equals(jobDefinition))
  orderby vacancyList.Title
  select new { vacancyList.Title, vacancyList.Location,
    vacancyList.InterviewerImnName };
foreach (var job in unfilteredJobs)
{
  jobRow = new TableRow();
  jobDetail = new TableCell();
  jobDetail.Text = job.Title;
  jobRow.Cells.Add(jobDetail);

  jobDetail = new TableCell();
  jobDetail.Text = job.Location;
  jobRow.Cells.Add(jobDetail);

  jobDetail = new TableCell();
  jobDetail.Text = job.InterviewerImnName;
  jobRow.Cells.Add(jobDetail);

  jobTable.Rows.Add(jobRow);
}
this.Controls.Add(jobTable);
```

13. On the **File** menu, click **Save All**.

Task 6: Deploy and test the visual Web Part

1. In the Solution Explorer window, right-click **Lab04**, and then click **Deploy**.

   Microsoft SharePoint builds, packages, and deploys the Web Parts.

2. On the **Start** menu, click **Internet Explorer**.
3. On the **Quick Launch** bar, click **Site Pages**.
4. Click **Recruitment**.
5. On the ribbon, click **Edit**.
6. On the ribbon, click the **Insert** tab.
7. On the ribbon, click **Web Part**.
8. In the **Categories** section, click **Custom**.
9. In the **Web Parts** section, click **JobDefinitions**.
10. Click **Add**.

    The Web Part is added to the page.

11. On the ribbon, click **Web Part**.
12. In the **Categories** section, click **Custom**.
13. In the **Web Parts** section, click **OpenPositions**.
14. Click **Add**.

    The Web Part is added to the page.

15. Click the **JobDefinitions Web Part** menu for the **JobDefinitions Web Part**.
16. Click **Edit Web Part**.
17. Click the **JobDefinitions Web Part** menu for the **JobDefinitions Web Part**.

18. Click **Connections**, click **Send Job To**, and then click **Open Positions**.
19. On the ribbon, click **Save and Close**.
20. In the **Job Definition Filter** drop-down list, click **Developer**.

    Developer jobs are displayed in the OpenPositions Web Part.
21. In the **Job Definition Filter** drop-down list, click **All Jobs**.

    All jobs are displayed in the OpenPositions Web Part.
22. Close all applications. You have now completed this lab.

Module 5

Lab Answer Key: Creating Event Receivers and Application Settings

Contents:

Lab: Creating Event Receivers and Web.Config Modifications

Log On to the Virtual Machine for This Lab

For this lab, use the available virtual machine environment. Before you begin the lab, log on to the **10175-LON-DEV-05** virtual machine as **Administrator**, with a password of **P@ssw0rd**.

Exercise 1: Creating List Event Receivers

▶ Task 1: Create an empty SharePoint project

1. On the **Start** menu, click **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.
2. On the **File** menu, point to **New**, and then click **Project**.
3. In the **Installed Templates** section, expand the **Visual C#** node, expand the **SharePoint** node and then click **2010**.
4. Click **Empty SharePoint Project**.
5. In the **Name** text box, type **Lab05**.
6. In the **Location** text box, type **E:\Student\Lab05\Starter**.
7. Click **OK**.

   The Microsoft® SharePoint® Customization Wizard appears.
8. In the **What local site do you want to use for debugging?** text box, type **http://sharepoint**.
9. In the **What is the trust level for this SharePoint** solution section, click the **Deploy as a farm solution** option.
10. Click **Finish**.

    The project is created.

▶ Task 2: Add event receivers to the InterviewOutcomes list

1. In the Solution Explorer window, right-click **Lab05**, point to **Add** and then click **New Item**.
2. Click **Event Receiver**.
3. In the **Name** text box, type **CheckJobs** and then click **Add**.

   The SharePoint Customization wizard appears.
4. In the **What type of event receiver do you want?** list, click **List Item Events**.
5. In the **What item should be the event source?** lists, click **InterviewOutcomes**.
6. In the **Handle the following events** checkbox list, check the **An item is being added** checkbox, and the **An item is being updated** checkbox.
7. Click **Finish**.

   Microsoft Visual Studio® adds an event receiver class with stub methods for the events you chose to handle.

▶ Task 3: Develop the event handlers in the event receiver class

1. Add the following function after the closing brace of the **ItemUpdating** event handler:

```
bool checkItem(SPItemEventProperties properties)
{
  string jobTitle =
    properties.AfterProperties["Title"].ToString();
  bool allowed = false;
  SPWeb jobDefWeb = null;
  SPList jobDefList;
  SPUser privilegedAccount =
    properties.Web.AllUsers[@"SHAREPOINT\SYSTEM"];
  SPUserToken privilegedToken = privilegedAccount.UserToken;
  try
  {
    using (SPSite elevatedSite =
      new SPSite(properties.Web.Url, privilegedToken))
    {
      using (SPWeb elevatedWeb = elevatedSite.OpenWeb())
      {
        jobDefWeb = elevatedWeb.Webs["JobData"];
        jobDefList = jobDefWeb.Lists["Job Definitions"];
        foreach (SPListItem item in jobDefList.Items)
        {
          if (item["Title"].ToString() == jobTitle)
          {
            allowed = true;
            break;
          }
        }
      }
    }
    return (allowed);
  }
  finally
  {
    jobDefWeb.Dispose();
  }
}
```

2. Replace the existing code in the **ItemAdding** event handler with the following code:

```
try
{
  bool allowed = checkItem( properties);
  if (!allowed)
  {
    properties.Status = SPEventReceiverStatus.CancelWithError;
    properties.ErrorMessage =
     "The job is not defined in the Job Definitions List";
    properties.Cancel = true;
  }
}
catch (Exception ex)
{
  properties.Status = SPEventReceiverStatus.CancelWithError;
  properties.ErrorMessage = ex.Message;
  properties.Cancel = true;
}
```

3. Replace the existing code in the **ItemUpdating** event handler with the following code:

```
try
{
  bool allowed = checkItem(properties);
  if (!allowed)
```

```
  {
    properties.Status = SPEventReceiverStatus.CancelWithError;
    properties.ErrorMessage =
      "The job is not defined in the Job Definitions List";
      properties.Cancel = true;
  }
}
catch (Exception ex)
{
  properties.Status = SPEventReceiverStatus.CancelWithError;
  properties.ErrorMessage = ex.Message;
  properties.Cancel = true;
}
```

4. On the **File** menu, click **Save All**.

▶ Task 4: Deploy and test the list event receiver

1. In the Solution Explorer window, right-click **Lab05** and then click **Deploy**.
2. Use Microsoft Internet Explorer® to browse to **http://sharepoint**.
3. On the **Quick Launch** bar, click **Outcomes**.
4. On the ribbon, click the **Items** tab.
5. On the ribbon, click **New Item**.
6. In the **Title** text box, type **Analyst/Developer**.
7. Click **Save**.

   An error dialog informs you that the job you entered in not defined in the Job Definitions list.
8. Close the dialog.
9. On the ribbon, click **New Item**.
10. In the **Title** text box, type **Developer**.
11. Click **Save**.

    The item is added to the list successfully.

Exercise 2: Creating Feature Receivers to Modify Web.Config

Task 1: Add a new feature and feature receiver to the project

1. In the Solution Explorer window, right-click **Features** and then click **Add Feature**.

   Microsoft Visual Studio® adds a new feature called Feature2.
2. In the Solution Explorer window, right-click **Feature2** and then click **Rename**.
3. Type **ControlProliferation** and press ENTER.
4. In the Solution Explorer window, double-click **ControlProliferation**.

   Visual Studio opens the Feature designer.
5. In the **Title** text box, type **Control Proliferation of Subsites**.
6. In the **Scope** drop-down list, click **Web**.
7. On the **File** menu, click **Save All**.
8. In the Solution Explorer window, right-click **ControlProliferation** and then click **Add Event Receiver**.

   Visual Studio adds a class to the project and opens it.
9. Select the three currently-commented out lines of code for the **FeatureActivated** event handler.
10. On the toolbar, click **Uncomment the selected lines**.
11. Select the three currently-commented out lines of code for the **FeatureDeactivating** event handler.
12. On the toolbar, click **Uncomment the selected lines**.

Task 2: Add code for modifying the Web.config file

1. Add the following statement to the empty line above the **namespace** declaration:

```
using Microsoft.SharePoint.Administration;
```

2. Add the following function after the closing brace of the **FeatureDeactivating** event handler:

```
void setProliferationFlag(bool status)
{
  SPWebApplication webApp =
    SPWebApplication.Lookup(new Uri("http://SharePoint"));
  try
{
  SPWebConfigModification mySetting = null;
  if (status)
  {
    mySetting = new SPWebConfigModification();
    mySetting.Path = "configuration/appSettings";
    mySetting.Name =
      "add [@key='preventProliferation'] [@value='1']";
    mySetting.Sequence = 0;
    mySetting.Owner = "Lab05Owner";
    mySetting.Type =
      SPWebConfigModification.SPWebConfigModificationType
       .EnsureChildNode;
    mySetting.Value =
      "<add key='preventProliferation' value='1' />";
    webApp.WebConfigModifications.Add(mySetting);
  }
  else
  {
```

```
    foreach (SPWebConfigModification modification in
      webApp.WebConfigModifications)
    {
      if (modification.Owner == "Lab05Owner")
      {
        modification.Value =
          "<add key='preventProliferation' value='0' />";
      }
    }
  }
  webApp.Update();
  webApp.Farm.Services.GetValue<SPWebService>()
    .ApplyWebConfigModifications();
  }
  catch
  {
  }
}
```

3. Add the following code between the opening and closing braces of the **FeatureActivated** event handler:

```
setProliferationFlag(true);
```

4. Add the following code between the opening and closing braces of the **FeatureDeactivating** event handler:

```
setProliferationFlag(false);
```

5. On the **File** menu, click **Save All**.

Exercise 3: Creating Web Event Receivers

Task 1: Add a Web event receiver

1. In the Solution Explorer window, right-click **Lab05**, point to **Add** and then click **New Item**.
2. Click **Event Receiver**.
3. In the **Name** text box, type **ControlSubsites**, and then click **Add**.

   The SharePoint Customization Wizard appears.
4. In the **What type of event receiver do you want?** list, click **Web Events**.
5. In the **Handle the following events** checkbox list, check the **A site is being provisioned** checkbox.
6. Click **Finish**.

   Visual Studio adds an event receiver class with stub methods for the event you chose to handle.

Task 2: Develop the Web event receiver to read the Web.config file

1. Add the following statements to the empty line above the **namespace** declaration:

```
using System.Configuration;
using System.Web;
```

2. Replace the existing code in the **WebAdding** event handler with the following code:

```
string preventProliferation = ConfigurationManager
  .AppSettings["preventProliferation"].ToString();
if (preventProliferation == "1")
{
  properties.ErrorMessage = "Sub webs are not allowed";
  properties.Status = SPEventReceiverStatus.CancelWithError;
  properties.Cancel = true;
  return;
}
base.WebAdding(properties);
```

3. On the **File** menu, click **Save All**.

Task 3: Deploy and test the Web event receiver

1. In the Solution Explorer window, right-click **Lab05** and then click **Deploy**.
2. Use Internet Explorer to browse to **http://sharepoint**.
3. On the **Site Actions** menu, click **Site Settings**.
4. In the **Site Actions** section of the page, click **Manage site features**.

   Your feature called Control Proliferation of Subsites is included in the feature list.
5. If the Control Proliferation of Subsites feature is not currently active, click **Activate**.
6. On the **Site Actions** menu, click **New Site**.
7. Click **Team Site**.
8. In the **Title** text box, type **Test**.
9. In the **URL name** text box, type **Test**.
10. Click **Create**.

    You are notified that subwebs are not allowed.

11. Close the **Error** dialog and close the **Create** dialog.
12. Use Internet Explorer to browse to **http://sharepoint**.
13. On the **Site Actions** menu, click **Site Settings**.
14. In the **Site Actions** section of the page, click **Manage site features**.
15. Click **Deactivate** for the **Control Proliferation of Subsites** feature.
16. Click **Deactivate this feature**.
17. On the **Site Actions** menu, click **New Site**.
18. Click **Team Site**.
19. In the **Title** text box, type **Test**.
20. In the **URL name** text box, type **Test**.
21. Click **Create**.

    Your site is created successfully.
22. Close all applications.

    You have now completed this lab.

Module 6

Lab Answer Key: Developing Solutions by Using Business Connectivity Services

Contents:

Lab: Building Business Connectivity Services Solutions

Log On to the Virtual Machine for This Lab

For this lab, you use the available virtual machine environment. Before you begin the lab, you must log on to the **10175-LON-DEV-06** virtual machine as **Administrator** using the password **P@ssw0rd**.

Exercise 1: Creating External Content Types and Lists by Using SharePoint Designer 2010

▶ Task 1: Explore the HRTrainingManagement database

1. On the **Start** menu, click **All Programs**, click **Microsoft SQL Server® 2008**, and then click **SQL Server Management Studio**.

   The Connect to Server dialog box appears.
2. In the **Server type** drop-down list, click **Database Engine**.
3. In the **Server name** drop-down list, type **SHAREPOINT\SharePoint**.
4. In the **Authentication** drop-down list, click **Windows Authentication**.
5. Click **Connect**.
6. In the Object Explorer window, expand **Databases**.
7. In the Object Explorer window, expand **HRTrainingManagement**.
8. In the Object Explorer window, expand **Database Diagrams**.
9. In the Object Explorer window, double-click **dbo.Schema**.
10. Review the database design. Note that the **TrainingSupplier** table is not related to any other table. Also note that the **TrainingObjects** table is related to the **Student** table with a many-to-many relationship implemented by joins to the **TrainingEvent** table.

▶ Task 2: Create a connection by using SharePoint Designer

1. Start Windows® Internet Explorer®.
2. On the **Site Actions** menu, click **Edit in SharePoint Designer**.

   The Microsoft® SharePoint® Designer starts.
3. In the Site Objects pane, click **External Content Types**.
4. On the ribbon, click **External Content Type**.
5. In the **External Content Type Information** section, next to **Name**, click **New external content type**.
6. Type **TrainingSuppliers**.
7. In the **External Content Type Operations** section, click **Click here to discover external data sources and define operations**.
8. Click **Add Connection**.
9. In the **Data Source Type** drop-down list, click **SQL Server**, and then click **OK**.
10. In the **Database Server** text box, type **SHAREPOINT\SharePoint**.

11. In the **Database Name** text box, type **HRTrainingManagement**.
12. Click the **Connect with User's Identity** option, and then click **OK**.

Task 3: Create an external content type

1. In the **Data Source Explorer** section, expand **HRTrainingManagement**.
2. Expand **Tables**.
3. Right-click **TrainingSupplier**, and then click **Create All Operations**.

   The All Operations dialog box appears.
4. Click **Next**.
5. In the **Data Source Elements** section, click **SupplierID**, but do not clear the check box.
6. Ensure that the **Map to Identifier** check box is selected.
7. Select the **Show In Picker** check box.
8. In the **Data Source Elements** section, click **Supplier**, but do not clear the check box.
9. Select the **Show In Picker** check box.
10. In the **Data Source Elements** section, click **Address**, but do not clear the check box.
11. Select the **Show In Picker** check box.
12. In the **Data Source Elements** section, click **PrimaryContact**, but do not clear the check box.
13. Select the **Show In Picker** check box.
14. In the **Data Source Elements** section, click **ContactTelephone**, but do not clear the check box.
15. Select the **Show In Picker** check box.
16. In the **Data Source Elements** section, click **ContactEmail**, but do not clear the check box.
17. Select the **Show In Picker** check box.
18. Click **Next**.
19. Click **Finish**.
20. On the **Quick Access** toolbar, click **Save**.

Task 4: Create a list and form for the new external content type

1. On the ribbon, click **Create Lists & Form**.

   The Create List and Form for New External Content Type dialog box appears.
2. In the **List Name** text box, type **Training Suppliers**, and then click **OK**.

Task 5: Secure and test the external list

1. On the **Start** menu, click **All Programs**, click **Microsoft SharePoint 2010 Products**, and then click **SharePoint 2010 Central Administration**.
2. In the **Application Management** section, click **Manage service applications**.
3. Click **Business Data Connectivity Service**.
4. Point to the **TrainingSuppliers** link, and then click the drop-down arrow that appears.
5. Click **Set Permissions**.
6. Click the **Browse** icon.

7. Click **All Users**.
8. Click **All Authenticated Users**.
9. Click **Add**, and then click **OK**.
10. Click **Add**.
11. Select the **Edit** check box.
12. Select the **Execute** check box.
13. Select the **Selectable In Clients** check box.
14. Select the **Set Permissions** check box, and then click **OK**.
15. Navigate to **http://sharepoint**.

    The Training Suppliers link appears on the Quick Launch bar.
16. Click **Training Suppliers**.
17. Click **Graphic Design Institute**.
18. On the ribbon, click **Edit Item**.
19. In the **Address** text box, change **Chicago** to **New York**.
20. Click **Save**.
21. On the ribbon, click **New Item**.
22. In the **Supplier** text box, type **Proseware, Inc**.
23. In the **Address** text box, type **789 Main Street, Los Angeles**.
24. In the **PrimaryContact** text box, type **Christiane Jensen**.
25. In the **ContactTelephone** text box, type **444-4444**.
26. In the **ContactEmail** text box, type **ChristianeJensen@proseware.com**.
27. Click **Save**.
28. Close all applications.

Exercise 2: Creating Business Data Catalog Models by Using Visual Studio 2010

Task 1: Create an empty SharePoint project

1. On the **Start** menu, click **All Programs**, open Microsoft Visual Studio® 2010 by clicking **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.
2. On the **File** menu, point to **New**, and then click **Project**.
3. In the **Installed Templates** section, expand the **Visual C#** node, expand the **SharePoint** node, and then click **2010**.
4. Click **Empty SharePoint Project**.
5. In the **Name** text box, type **Lab06**.
6. In the **Location** text box, type **E:\Student\Lab06\Starter**, and then click **OK**.

   The SharePoint Customization Wizard appears.
7. In the **What local site do you want to use for debugging?** text box, type **http://sharepoint**.
8. In the **What is the trust level for this SharePoint solution?** section, click the **Deploy as a farm solution** option.
9. Click **Finish**.

   The project is created.

Task 2: Add and configure a Business Data Connectivity model

1. In the Solution Explorer window, right-click **Lab06**, point to **Add**, and then click **New Item**.
2. Click **Business Data Connectivity Model**.
3. In the **Name** text box, type **TrainingEvent**, and then click **Add**.
4. In the Solution Explorer window, right-click **Entity1.cs**, and then click **Rename**.
5. Type **TrainingEvent.cs**, and then press ENTER.
6. In the **Microsoft Visual Studio** dialog box that prompts you to rename all references to Entity1, click **Yes**. If you are prompted to reload the file, click **Yes**.
7. Double-click **TrainingEvent.cs**.

   The TrainingEvent.cs code file appears.
8. Replace the existing property declarations in the class with the following:

```
public Int32 TrainingEventID { get; set; }
public DateTime EventDate { get; set; }
public string Status { get; set; }
public string LoginName { get; set; }
public string Title { get; set; }
public string EventType { get; set; }
public string Description { get; set; }
```

9. In the Solution Explorer window, double-click **TrainingEvent.bdcm**.
10. On the design surface, click **Entity1**.
11. On the **View** menu, click **Properties Window**.
12. In the Properties window, click **Name**.
13. Type **TrainingEventEntity**, and press ENTER.

14. On the design surface, click **Identifier1**.
15. In the Properties window, click **Name**.
16. Type **TrainingEventID**, and press ENTER.
17. In the Properties window, click **Type Name**.
18. In the drop-down list for the **Type Name** property, click **System.Int32**.
19. On the **File** menu, click **Save All**.

▶ Task 3: Define type descriptors for the ReadItem method

1. On the design surface, click **ReadItem**.
2. In the BDC Method Details pane, in the **Parameters** section below **ReadItem**, click **Identifier1** (in the **id** row). Then, click the drop-down arrow that appears and click **<Edit>**.
3. In the Properties window, click the **Name** property, type **TrainingEventID**, and then press ENTER.
4. In the Properties window, click **Type Name**.
5. In the drop-down list for the **Type Name** property, click **Int32**.
6. In the BDC Method Details pane, in the **Parameters** section below **ReadItem**, click **Entity1** (in the **returnParameter** row). Then, click the drop-down arrow that appears and click **<Edit>**.
7. In the Properties window, click the **Name** property, type **TrainingEvent**, and then press ENTER.
8. In the BDC Explorer window, expand the currently selected **TrainingEvent** node.
9. Beneath the **TrainingEvent** node, click **Identifier1**.
10. In the Properties window, click the **Name** property, type **TrainingEventID**, and then press ENTER.
11. In the Properties window, click **Type Name**.
12. In the drop-down list for the **Type Name** property, click **Int32**.
13. In the Properties window, click **Read-only**.
14. In the drop-down list for the **Read-only** property, click **True**.
15. In the BDC Explorer window, click **Message**.
16. In the Properties window, click the **Name** property, type **EventDate**, and then press ENTER.
17. In the Properties window, click **Type Name**.
18. In the drop-down list for the **Type Name** property, click **DateTime**.
19. In the BDC Explorer window, right-click **TrainingEvent**, and then click **Add Type Descriptor**.
20. In the Properties window, click the **Name** property, type **Status**, and then press ENTER.
21. In the BDC Explorer window, right-click **TrainingEvent**, and then click **Add Type Descriptor**.
22. In the Properties window, click the **Name** property, type **LoginName**, and then press ENTER.
23. In the BDC Explorer window, right-click **TrainingEvent**, and then click **Add Type Descriptor**.
24. In the Properties window, click the **Name** property, type **Title**, and then press ENTER.
25. In the BDC Explorer window, right-click **TrainingEvent**, and then click **Add Type Descriptor**.
26. In the Properties window, click the **Name** property, type **EventType**, and then press ENTER.
27. In the BDC Explorer window, right-click **TrainingEvent**, and then click **Add Type Descriptor**.

28. In the Properties window, click the **Name** property, type **Description**, and then press ENTER.
29. On the **File** menu, click **Save All**.

Task 4: Define type descriptors for the ReadList method

1. In the BDC Method Details pane, in the **Parameters** section below **ReadList**, click **Entity1List** (in the **returnParameter** row). Then, click the drop-down arrow that appears and click **<Edit>**.
2. In the Properties window, click the **Name** property, type **TrainingEventList**, and then press ENTER.
3. In the BDC Explorer window, expand the **TrainingEventList** node.
4. Right-click **Entity1**, and then click **Delete**.
5. In the BDC Explorer window, below the **returnParameter** for the **ReadItem** node, right-click **TrainingEvent**, and then click **Copy**.
6. In the BDC Explorer window, below the **returnParameter** for the **ReadList** node, right-click **TrainingEventList**, and then click **Paste**.
7. On the **File** menu, click **Save All**.

Task 5: Create and configure data operation methods

1. In the BDC Method Details pane, collapse the **ReadList** and **ReadItem** nodes.
2. Click <**Add a Method>**, and then in the drop-down list, click **Create Creator Method**.
3. In the **Parameters** section below **Create**, click **NewTrainingEventEntity** in the **Type Descriptor** column for the **newTrainingEventEntity** row.
4. Click the drop-down arrow, and then click **<Edit>**.
5. In the BDC Explorer window, expand the selected **NewTrainingEventEntity** node, and then click **TrainingEventID**.
6. In the Properties window, click **Read-only**, click the drop-down arrow, and then click **False**.
7. Collapse the **Create** node.
8. Click <**Add a Method>**, and then in the drop-down list, click **Create Deleter Method**.
9. Collapse the **Delete** node.
10. Click <**Add a Method>**, and then in the drop-down list, click **Create Updater Method**.
11. In the **Parameters** section below **Update**, click **TrainingEventEntity** in the **Type Descriptor** column for the **trainingEventEntity** row.
12. Click the drop-down arrow, and then click **<Edit>**.
13. In the BDC Explorer window, expand the selected **TrainingEventEntity** node, and then click **TrainingEventID**.
14. In the Properties window, click **Read-only**, click the drop-down arrow, and then click **False**.
15. On the **File** menu, click **Save All**.

Task 6: Implement the methods for the TrainingEventEntity entity

1. On the design surface, double-click **ReadItem**.

   The code file for the **TrainingEventEntityService.cs** class appears.

2. Add the following using statements above the namespace declaration:

```
using System.Data;
using System.Data.Sql;
using System.Data.SqlClient;
using System.Data.SqlTypes;
```

3. Place the cursor *after* the *opening* brace of the class definition that reads **public class TrainingEventEntityService**, and press ENTER.
4. Add the following function to the space you have just created:

```
static SqlConnection getSqlConnection()
{
  SqlConnection sqlConn = new SqlConnection(
    "Integrated Security=SSPI;Persist Security Info=False;"
    + "Initial Catalog=HRTrainingManagement;"
    +  @"Data Source=SHAREPOINT\SharePoint");
  return (sqlConn);
}
```

5. Replace the contents of the **ReadItem** method with the following code:

```
SqlConnection thisConn = null;
TrainingEvent evt = null;
try
{
  evt = new TrainingEvent();
  thisConn = getSqlConnection();
  thisConn.Open();
  SqlCommand thisCmd = new SqlCommand();
  thisCmd.CommandText = "SELECT e.TrainingEventID, s.LoginName,"
  + " t.Title, t.EventType, t.Description, e.EventDate, e.Status"
  + " FROM Student s"
  + " INNER JOIN TrainingEvent e ON s.StudentID = e.StudentID"
  + " INNER JOIN TrainingObjects t ON e.TrainingID = t.TrainingID"
  + " WHERE e.TrainingEventID = " + id.ToString();
  thisCmd.Connection = thisConn;
  SqlDataReader thisReader =
    thisCmd.ExecuteReader(CommandBehavior.CloseConnection);
  if (thisReader.Read())
  {
    evt.TrainingEventID = id;
    evt.LoginName = thisReader[1].ToString();
    evt.Title = thisReader[2].ToString();
    evt.EventType = thisReader[3].ToString();
    evt.Description = thisReader[4].ToString();
    evt.EventDate = DateTime.Parse(thisReader[5].ToString());
    evt.Status = thisReader[6].ToString();
  }
  else
  {
    evt.TrainingEventID = -1;
    evt.LoginName = "Data Not Found";
    evt.Title = "Data Not Found";
    evt.EventType = "Data Not Found";
    evt.Description = "Data Not Found";
    evt.EventDate = DateTime.MinValue;
    evt.Status = "Data Not Found";
  }
  thisReader.Close();
  return (evt);
}
catch
{
```

```
  evt.TrainingEventID = -1;
  evt.LoginName = "Data Not Found";
  evt.Title = "Data Not Found";
  evt.EventType = "Data Not Found";
  evt.Description = "Data Not Found";
  evt.EventDate = DateTime.MinValue;
  evt.Status = "Data Not Found";
  return (evt);
}
finally
{
    thisConn.Dispose();
}
```

6. Replace the contents of the **ReadList** method with the following code:

```
SqlConnection thisConn = null;
List<TrainingEvent> allEvents;
try
{
  thisConn = getSqlConnection();
  allEvents = new List<TrainingEvent>();
  thisConn.Open();
  SqlCommand thisCommand = new SqlCommand();
  thisCommand.Connection = thisConn;
  thisCommand.CommandText = "SELECT e.TrainingEventID, LoginName,"
  + " t.Title, t.EventType, t.Description, e.EventDate, e.Status"
  + " FROM Student s"
  + " INNER JOIN TrainingEvent e ON s.StudentID = e.StudentID"
  + " INNER JOIN TrainingObjects t ON e.TrainingID = t.TrainingID";
  SqlDataReader thisReader =
    thisCommand.ExecuteReader(CommandBehavior.CloseConnection);
  while (thisReader.Read())
  {
    TrainingEvent evt = new TrainingEvent();
    evt.TrainingEventID = int.Parse(thisReader[0].ToString());
    evt.LoginName = thisReader[1].ToString();
    evt.Title = thisReader[2].ToString();
    evt.EventType = thisReader[3].ToString();
    evt.Description = thisReader[4].ToString();
    evt.EventDate = DateTime.Parse(thisReader[5].ToString());
    evt.Status = thisReader[6].ToString();
    allEvents.Add(evt);
  }
  thisReader.Close();
  TrainingEvent[] eventList = new TrainingEvent[allEvents.Count];
  for (int evtCounter = 0;
    evtCounter <= allEvents.Count - 1;
    evtCounter++)
  {
    eventList[evtCounter] = allEvents[evtCounter];
  }
  return (eventList);
}
catch (Exception ex)
{
  TrainingEvent[] errEventList = new TrainingEvent[1];
  TrainingEvent errEvt = new TrainingEvent();
  errEvt.TrainingEventID = -1;
  errEvt.LoginName = ex.Message;
  errEvt.Title = ex.Message;
  errEvt.EventType = ex.Message;
  errEvt.Description = ex.Message;
```

```
  errEvt.EventDate = DateTime.MinValue;
  errEvt.Status = ex.Message;
  errEventList[0] = errEvt;
  return (errEventList);
}
finally
{
    thisConn.Dispose();
}
```

7. Replace the contents of the **Create** method with the following code:

```
SqlConnection thisConn = null;
try
{
  thisConn = getSqlConnection();
  thisConn.Open();
  string studentName = newTrainingEventEntity.LoginName;
  string trainingTitle = newTrainingEventEntity.Title;
  string trainingType = newTrainingEventEntity.EventType;
  string trainingDescription = newTrainingEventEntity.Description;
  DateTime trainingDate = newTrainingEventEntity.EventDate;
  string trainingStatus = newTrainingEventEntity.Status;
  int studentID = 0;
  SqlCommand studentCmd = new SqlCommand();
  studentCmd.Connection = thisConn;
  studentCmd.CommandText = "SELECT StudentID"
    + " FROM Student"
    + " WHERE LoginName='" + studentName + "'";
  SqlDataReader studentReader =
    studentCmd.ExecuteReader(CommandBehavior.Default);
  if (studentReader.Read())
  {
    studentID = int.Parse(studentReader[0].ToString());
    studentReader.Close();
  }
  else
  {
    studentReader.Close();
    SqlCommand addStudentCommand = new SqlCommand();
    addStudentCommand.Connection = thisConn;
    addStudentCommand.CommandText =
      "INSERT Student(LoginName) VALUES('" + studentName + "')";
    addStudentCommand.ExecuteNonQuery();
    SqlCommand getNewStudentCmd = new SqlCommand();
    getNewStudentCmd.Connection = thisConn;
    getNewStudentCmd.CommandText = "SELECT StudentID"
      + " FROM Student"
      + " WHERE LoginName = '" + studentName + "'";
    SqlDataReader getNewStudentReader =
      getNewStudentCmd.ExecuteReader(CommandBehavior.Default);
    getNewStudentReader.Read();
    studentID = int.Parse(getNewStudentReader[0].ToString());
    getNewStudentReader.Close();
  }
  int trainingID = 0;
  SqlCommand trainingCmd = new SqlCommand();
  trainingCmd.Connection = thisConn;
  trainingCmd.CommandText = "SELECT TrainingID"
    + " FROM TrainingObjects"
    + " WHERE Title = '" + trainingTitle + "'"
    + " AND EventType = '" + trainingType + "'"
    + " AND Description = '" + trainingDescription + "'";
```

```
  SqlDataReader trainingReader =
    trainingCmd.ExecuteReader(CommandBehavior.Default);
  if (trainingReader.Read())
  {
    trainingID = int.Parse(trainingReader[0].ToString());
    trainingReader.Close();
  }
  else
  {
    trainingReader.Close();
    SqlCommand addTrainingCommand = new SqlCommand();
    addTrainingCommand.Connection = thisConn;
    addTrainingCommand.CommandText =
      "INSERT TrainingObjects(Title, EventType, Description)"
      + " VALUES('" + trainingTitle + "','"
      + trainingType + "','"
      + trainingDescription + "')";
    addTrainingCommand.ExecuteNonQuery();
    SqlCommand getNewTrainingCmd = new SqlCommand();
    getNewTrainingCmd.Connection = thisConn;
    getNewTrainingCmd.CommandText = "SELECT TrainingID"
      + " FROM TrainingObjects"
      + " WHERE Title = '" + trainingTitle + "'"
      + " AND EventType = '" + trainingType + "'"
      + " AND Description = '" + trainingDescription + "'";
    SqlDataReader getNewTrainingReader =
      getNewTrainingCmd.ExecuteReader(CommandBehavior.Default);
    getNewTrainingReader.Read();
    trainingID = int.Parse(getNewTrainingReader[0].ToString());
    getNewTrainingReader.Close();
  }
  SqlCommand insertEventCommand = new SqlCommand();
  insertEventCommand.Connection = thisConn;
  insertEventCommand.CommandText = "INSERT TrainingEvent"
    + "(StudentID, TrainingID, EventDate, Status) VALUES("
    + studentID
    + ", " + trainingID
    + ", '" + trainingDate.ToShortDateString() + "'"
    + ", '" + trainingStatus + "')";
  insertEventCommand.ExecuteNonQuery();
  return (newTrainingEventEntity);
}
finally
{
  thisConn.Dispose();
}
```

8. Replace the contents of the **Delete** method with the following code:

```
SqlConnection thisConn = null;
try
{
  thisConn = getSqlConnection();
  thisConn.Open();
  SqlCommand thisCommand = new SqlCommand();
  thisCommand.Connection = thisConn;
  thisCommand.CommandText =
    "DELETE TrainingEvent WHERE TrainingEventID = "
    + trainingEventID.ToString();
  thisCommand.ExecuteNonQuery();
}
finally
{
```

```
  thisConn.Dispose();
}
```

9. Replace the contents of the **Update** method with the following code:

```
SqlConnection thisConn = null;
try
{
  thisConn = getSqlConnection();
  thisConn.Open();
  int trainingEventID = trainingEventEntity.TrainingEventID;
  string studentName = trainingEventEntity.LoginName;
  string trainingTitle = trainingEventEntity.Title;
  string trainingType = trainingEventEntity.EventType;
  string trainingDescription = trainingEventEntity.Description;
  DateTime trainingDate = trainingEventEntity.EventDate;
  string trainingStatus = trainingEventEntity.Status;
  int studentID = 0;
  SqlCommand studentCmd = new SqlCommand();
  studentCmd.Connection = thisConn;
  studentCmd.CommandText = "SELECT s.StudentID, s.LoginName"
   + " FROM Student s"
   + " INNER JOIN TrainingEvent e ON s.StudentID = e.StudentID"
   + " WHERE e.TrainingEventID = " + trainingEventID.ToString();
  SqlDataReader studentReader =
    studentCmd.ExecuteReader(CommandBehavior.Default);
  studentReader.Read();
  if (studentReader[1].ToString() == studentName)
  {
    studentID = int.Parse(studentReader[0].ToString());
    studentReader.Close();
  }
  else
  {
    studentReader.Close();
    SqlCommand changeStudentCmd = new SqlCommand();
    changeStudentCmd.Connection = thisConn;
    changeStudentCmd.CommandText = "SELECT StudentID"
      + " FROM Student"
      + " WHERE LoginName = '" + studentName + "'";
    SqlDataReader changeStudentReader =
    changeStudentCmd.ExecuteReader(CommandBehavior.Default);
    if (changeStudentReader.Read())
    {
      studentID = int.Parse(changeStudentReader[0].ToString());
      changeStudentReader.Close();
    }
    else
    {
      changeStudentReader.Close();
      SqlCommand addStudentCommand = new SqlCommand();
      addStudentCommand.Connection = thisConn;
      addStudentCommand.CommandText =
       "INSERT Student(LoginName) VALUES('" + studentName + "')";
      addStudentCommand.ExecuteNonQuery();
      SqlCommand getNewStudentCmd = new SqlCommand();
      getNewStudentCmd.Connection = thisConn;
      getNewStudentCmd.CommandText = "SELECT StudentID"
        + " FROM Student"
        + " WHERE LoginName = '" + studentName + "'";
      SqlDataReader getNewStudentReader =
        getNewStudentCmd.ExecuteReader(CommandBehavior.Default);
      getNewStudentReader.Read();
```

```
    studentID = int.Parse(getNewStudentReader[0].ToString());
    getNewStudentReader.Close();
  }
}
int trainingID = 0;
SqlCommand trainingCmd = new SqlCommand();
studentCmd.Connection = thisConn;
studentCmd.CommandText =
  "SELECT t.TrainingID, t.Title, t.EventType, t.Description"
  + " FROM TrainingObjects t"
  + " INNER JOIN TrainingEvent e ON t.TrainingID = e.TrainingID"
  + " WHERE e.TrainingEventID = " + trainingEventID.ToString();
SqlDataReader trainingReader =
studentCmd.ExecuteReader(CommandBehavior.Default);
trainingReader.Read();
if ((trainingReader[1].ToString() == trainingTitle)
  && (trainingReader[2].ToString() == trainingType)
  && (trainingReader[3].ToString() == trainingDescription))
{
  trainingID = int.Parse(trainingReader[0].ToString());
  trainingReader.Close();
}
else
{
  trainingReader.Close();
  SqlCommand changeTrainingCmd = new SqlCommand();
  changeTrainingCmd.Connection = thisConn;
  changeTrainingCmd.CommandText = "SELECT TrainingID"
  + " FROM TrainingObjects"
  + " WHERE Title = '" + trainingTitle + "'"
  + " AND EventType = '" + trainingType + "'"
  + " AND Description = '" + trainingDescription + "'";
  SqlDataReader changeTrainingReader =
    changeTrainingCmd.ExecuteReader(CommandBehavior.Default);
  if (changeTrainingReader.Read())
  {
    trainingID = int.Parse(changeTrainingReader[0].ToString());
    changeTrainingReader.Close();
  }
  else
  {
    changeTrainingReader.Close();
    SqlCommand addTrainingCommand = new SqlCommand();
    addTrainingCommand.Connection = thisConn;
    addTrainingCommand.CommandText = "INSERT TrainingObjects("
      + "Title, EventType, Description) VALUES('"
      + trainingTitle + "','"
      + trainingType + "','"
      + trainingDescription + "')";
    addTrainingCommand.ExecuteNonQuery();
    SqlCommand getNewTrainingCmd = new SqlCommand();
    getNewTrainingCmd.Connection = thisConn;
    getNewTrainingCmd.CommandText = "SELECT TrainingID"
      + " FROM TrainingObjects"
      + " WHERE Title = '" + trainingTitle + "'"
      + " AND EventType = '" + trainingType + "'"
      + " AND Description = '" + trainingDescription + "'";
    SqlDataReader getNewTrainingReader =
      getNewTrainingCmd.ExecuteReader(CommandBehavior.Default);
    getNewTrainingReader.Read();
    trainingID = int.Parse(getNewTrainingReader[0].ToString());
    getNewTrainingReader.Close();
  }
}
```

```
  SqlCommand updateEventCommand = new SqlCommand();
  updateEventCommand.Connection = thisConn;
  updateEventCommand.CommandText = "UPDATE TrainingEvent"
    + " SET StudentID=" + studentID
    + ", TrainingID=" + trainingID
    + ", EventDate='" + trainingDate.ToShortDateString() + "'"
    + ", Status='" + trainingStatus + "'"
    + " WHERE TrainingEventID=" + trainingEventID;
  updateEventCommand.ExecuteNonQuery();
}
finally
{
  thisConn.Dispose();
}
```

10. On the **File** menu, click **Save All**.

Task 7: Deploy and test the solution

1. In the Solution Explorer window, right-click **Lab06**, and then click **Deploy**.
2. On the **Start** menu, click **All Programs**, click **Microsoft SharePoint 2010 Products**, and then click **SharePoint 2010 Central Administration**.
3. In the **Application Management** section, click **Manage service applications**.
4. Click **Business Data Connectivity Service**.
5. Point to the **TrainingEventEntity** link, and then click the drop-down arrow that appears.
6. Click **Set Permissions**.
7. Click the **Browse** icon.
8. Click **All Users**.
9. Click **All Authenticated Users**.
10. Click **Add**, and then click **OK**.
11. Click **Add**.
12. Select the **Edit** check box.
13. Select the **Execute** check box.
14. Select the **Selectable In Clients** check box.
15. Select the **Set Permissions** check box, and then click **OK**.
16. Navigate to **http://sharepoint**.
17. On the **Site Actions menu**, click **More Options**.

    A list of templates appears.
18. Click **External List**.
19. Click **Create**.
20. In the **Name** text box, type **Training Events**.
21. In the **Data source configuration** section, click **Select External Content Type**.

    The External Content Type Picker dialog box appears.
22. Click **TrainingEvent**, and then click **OK**.

23. Click **Create**.

    The empty list appears.
24. On the ribbon, click the **Items** tab.
25. Click **New Item**.
26. In the **TrainingEventID** text box, type **1**.
27. In the **EventDate** text box, type **9/9/2011**.
28. In the **LoginName** text box, type **SHAREPOINT\LauraG**.
29. In the **Status** text box, type **Not Started**.
30. In the **Title** text box, type **SharePoint 101**.
31. In the **Description** text box, type **End User Course for SharePoint 2010**.
32. In the **EventType** text box, type **Instructor-Led Training**.
33. Click **Save**.

    The new item is added to the list.
34. Click the item that has just been added to the list.
35. On the ribbon, click **Edit Item**.
36. In the **EventType** text box, type **E-Learning**.
37. Click **Save**.
38. Click the list item.
39. On the ribbon, click **Delete Item**, and then click **OK**.

    The item is deleted.
40. Close all applications. You have now completed this lab.

Module 7

Lab Answer Key: Developing SharePoint 2010 Workflows

Contents:

Lab: Creating Workflows for SharePoint 2010

Log On to the Virtual Machine for This Lab

For this lab, use the available virtual machine environment. Before you begin the lab, log on to the **10175-LON-DEV-07** virtual machine as **Administrator**, with the password **P@ssw0rd**.

Exercise 1: Creating Workflows by Using SharePoint Designer

Task 1: Open the site in SharePoint Designer

1. Use Windows® Internet Explorer® to browse to **http://sharepoint**.
2. On the **Site Actions** menu, click **Edit in SharePoint Designer**.
3. In the Site Objects pane, click **Lists and Libraries**.
4. Click **Employment Contracts**.
5. In the **Settings** section, click **Require content approval for submitted items**.
6. On the **Quick Access Toolbar**, click **Save**.

Task 2: Create a new workflow with initiation parameters and variables

1. In the **Workflows** section, click **New**.
2. In the **Name** text box, type **Approve Contract**.
3. Click **OK**.
4. On the ribbon, click **Initiation Form Parameters**.
5. Click **Add**.
6. In the **Field name** text box, type **ApproverEmail**.
7. In the **Information type** drop-down list, click **Person or Group**.
8. Click **Next**.
9. Click **Finish**.
10. Click **Add**.
11. In the **Field name** text box, type **MaxSalary**.
12. In the **Information type** drop-down list, click **Number (1, 1.0, 100)**.
13. Click **Next**.
14. In the **Default value** text box, type **120000**.
15. Click **Finish**.
16. Click **OK**.
17. On the ribbon, click **Local Variables**.
18. Click **Add**.
19. In the **Name** text box, type **Salary**.
20. In the **Type** drop-down list, click **Number**.
21. Click **OK**.
22. Click **OK**.

Task 3: Add actions and decision-making to the workflow

1. On the ribbon, click **Action**, and then click **Set Workflow Variable**.
2. Click the **Workflow variable** link, and then click **Variable: Salary**.
3. Click the **value** link, and then click the **Define workflow lookup** button.
4. In the **Data source** drop-down list, click **Current Item**.
5. In the **Field from source** drop-down list, click **Salary**.
6. Click **OK**.
7. On the ribbon, click **Condition**, and then click **If any value equals value**.
8. Click the first **value** link, and then click the **Define workflow lookup** button.
9. In the **Data source** drop-down list, click **Workflow variables and parameters**.
10. In the **Field from source** drop-down list, click **Variable: Salary**.
11. Click **OK**.
12. Click the **equals** link, and then click **is greater than**.
13. Click the second **value** link, and then click the **Define workflow lookup** button.
14. In the **Data source** drop-down list, click **Workflow variables and parameters**.
15. In the **Field from source** drop-down list, click **Parameter: MaxSalary**.
16. Click **OK**.
17. Click **(Start typing or use the Insert group in the Ribbon)**.
18. On the ribbon, click **Action**, and then click **Set Field in Current Item**.
19. Click the **field** link, and then click **Review Comments**.
20. Click the **value** link, and then type **Auto-rejected: Salary breaches the current salary cap**. Then press ENTER.
21. On the ribbon, click **Action**, and then click **Set Content Approval Status**.
22. Click the **this status** link, and then click **Rejected**.
23. Click the **comments** link, and then type **Rejected by workflow**. Then press ENTER.
24. On the ribbon, click **Else-If Branch**.
25. On the ribbon, click **Action**, and then click **Set Content Approval Status**.
26. Click the **this status** link, and then click **Pending**.
27. Click the **comments** link, and then type **Awaiting review**. Then press ENTER.
28. On the ribbon, click **Action**, and then click **Set Field in Current Item**.
29. Click the **field** link, and then click **Reviewer**.
30. Click the **value** link.
31. Click **Workflow Lookup for a User**, and then click **Add**.
32. In the **Data source** drop-down list, click **Workflow variables and parameters**.
33. In the **Field from source** drop-down list, click **Parameter: ApproverEmail**.
34. In the **Return field as** drop-down list, click **Email Address**, then click **OK** and then click **OK** again.

35. On the **Quick Access Toolbar**, click **Save**.
36. On the ribbon, click **Publish**.
37. Close SharePoint Designer.

Task 4: Test the workflow

1. Use Internet Explorer to browse to **http://sharepoint**.
2. In the **Quick Launch** bar, click **Employment Contracts**.
3. On the ribbon, click the **Documents** tab.
4. On the ribbon, click **Upload Document**.
5. Click **Browse**, and browse to the **E:\Student\Lab07\Documents** folder.
6. Click **Employment Contract Bobby Moore.docx**, and then click **Open**.
7. Click **OK**.
8. In the **Title** text box, type **Bobby Moore**.
9. In the **Salary** text box, type **130000**.
10. Click **Save**.
11. In the list, point to **Employment Contract Bobby Moore**, click the drop-down arrow, and then click **Workflows**.
12. Click **Approve Contract**.
13. In the **ApproverEmail** text box, type **krishnas@sharepoint.com**.
14. Click **Check Names**.
15. Click **Start**.

    The workflow runs, and sets the Review comments and Approval Status columns appropriately for a rejected contract.
16. In the list, point to **Employment Contract Bobby Moore**, click the drop-down arrow, and then click **Edit Properties**.
17. Edit the **Salary** text box so that it contains **125000**.
18. Delete the contents of the **Review Comments** text box.
19. Click **Save**.
20. In the list, point to **Employment Contract Bobby Moore**, click the drop-down arrow, and then click **Workflows**.
21. Click **Approve Contract**.
22. In the **ApproverEmail** text box, type **krishnas@sharepoint.com**.
23. Click **Check Names**.
24. Edit the **MaxSalary** text box so that it contains **125,500**.
25. Click **Start**.

    The workflow runs, and sets the Approval Status to pending and the Reviewer to krishnas@sharepoint.com.
26. Close all applications.

Exercise 2: Creating a Sequential Workflow by Using Visual Studio 2010

Task 1: Create an empty SharePoint project

1. On the **Start** menu, click **All Programs**, click **Microsoft Visual Studio 2010** and then click **Microsoft Visual Studio 2010**.
2. On the **File** menu, point to **New**, and then click **Project**.
3. In the **Installed Templates** section, expand the **Visual C#** node, expand the **SharePoint** node, and then click **2010**.
4. Click **Empty SharePoint Project**.
5. In the **Name** text box, type **Lab07**.
6. In the **Location** text box, type **E:\Student\Lab07\Starter**.
7. Click **OK**.

   The SharePoint Customization Wizard appears.
8. In the **What local site do you want to use for debugging?** text box, type **http://sharepoint**.
9. In the **What is the trust level for this SharePoint** solution section, click the **Deploy as a farm solution** option.
10. Click **Finish**.

    The project is created.

Task 2: Add a sequential workflow

1. In the Solution Explorer window, right-click **Lab07**, point to **Add** and then click **New Item**.
2. Click **Sequential Workflow**.
3. In the **Name** text box, type **reviewContracts** and then click **Add**.

   The SharePoint Customization Wizard appears.
4. In the **What is the name of the workflow?** text box, type **reviewContracts**.
5. Click **List Workflow**.
6. Click **Next**.
7. In the **The library or list to associate your workflow with** drop-down list, click **Employment Contracts**.
8. Click **Next**.
9. Be sure the **A user manually starts the workflow** checkbox is checked.
10. Be sure the **The workflow starts automatically when an item is created** checkbox is checked.
11. Be sure the **The workflow starts automatically when an item is changed** checkbox is *not* checked.
12. Click **Finish**.
13. On the **View** menu, click **Toolbox**.
14. Click the **Auto Hide** pin to pin the toolbox open.
15. In the toolbox, expand the **Windows Workflow v3.0** group.
16. Drag a **While** item from the toolbox, and drop it beneath the **onWorkflowActivated1** item on the workflow design surface.

17. In the toolbox, expand the **SharePoint Workflow** group.
18. Drag an **OnWorkflowItemChanged** item from the toolbox, and drop it on the **Drop an Activity Here** text in the **whileActivity1** item.
19. Click the **onWorkflowActivated1** item on the workflow design surface.
20. In the Properties window, click **Invoked**.
21. Type **onWorkflowActivated** and then press ENTER.

    Microsoft Visual Studio® displays the code file.
22. Click the **reviewContracts.cs [Design]** tab.
23. Click the **whileActivity1** item on the workflow design surface. (Be sure you click the **whileActivity1** item, and not the **onWorkflowItemChanged1** item.)
24. In the Properties window, click **Conditions**, and then click the drop-down arrow next to **(None)**. Then click **Code Condition**.
25. Expand the **Condition** property, and then click the **Condition** row that appears.
26. Type **isWorkflowPending**, and then press ENTER.

    Visual Studio displays the code file.
27. Click the **reviewContracts.cs [Design]** tab.
28. Click the **onWorkflowItemChanged1** item on the workflow design surface. (Be sure you click the **onWorkflowItemChanged1** item, and not the **whileActivity1** item.)
29. In the Properties window, click **CorrelationToken**, click the drop-down arrow, and then click **workflowToken**.
30. In the Properties window, click **Invoked**.
31. Type **onWorkflowItemChanged** and press ENTER.
32. On the **File** menu, click **Save All**.

▶ Task 3: Develop the sequential workflow

1. On the line above the **onWorkflowActivated** method, add the following code:

```
bool wfPending = true;
```

2. Create an empty line after the code you have just typed.
3. Add the following code to the space you have just created:

```
void checkDocStatus()
{
   if(workflowProperties.Item["Contract Status"].ToString()
      == "Review Complete")
  {
    wfPending = false;
  }
}
```

4. Add the following code between the opening and closing braces of the **onWorkflowActivated** method:

```
checkDocStatus();
```

5. Add the following code between the opening and closing braces of the **isWorkflowPending** method:

```
e.Result = wfPending;
```

6. Add the following code between the opening and closing braces of the **onWorkflowItemChanged** method:

```
checkDocStatus();
```

Task 4: Test the sequential workflow

1. In the Solution Explorer window, right-click **Lab07** and then click **Deploy**.
2. Use Internet Explorer to browse to **http://sharepoint**.
3. In the **Quick Launch** bar, click **Employment Contracts**.
4. On the ribbon, click the **Documents** tab.
5. Click **Upload Document**.
6. Click **Browse**, and browse to the **E:\Student\Lab07\Documents** folder.
7. Click **Employment Contract Paula Bento.docx**, and then click **Open**.
8. Click **OK**.
9. In the **Title** text box, type **Paula Bento**.
10. In the **Contract Status** drop-down list, click **Review Needed**.
11. In the **Salary** text box, type **120000**.
12. Click **Save**.
13. Scroll to the right of the list, and note that the **ReviewContracts** workflow is **In Progress**.
14. In the list, point to **Employment Contract Paula Bento**, click the drop-down arrow, and then click **Edit Properties**.
15. In the **Contract Status** drop-down list, click **Review Complete**.
16. Click **Save**.
17. In the **Quick Launch** bar, click **Employment Contracts** to refresh the view of the list.
18. Scroll to the right of the list, and note that the **ReviewContracts** workflow is **Completed**.
19. Close all applications.

    You have now completed this lab.

Module 8

Lab Answer Key: Working with Client-Based APIs for SharePoint 2010

Contents:

Lab: Developing .NET Applications by Using the SharePoint Client Object Model

Log On to the Virtual Machine for This Lab

For this lab, you use the available virtual machine environment. Before you begin the lab, you must log on to the **10175-LON-DEV-08** virtual machine as **Administrator** using the password **P@ssw0rd**.

Exercise 1: Creating a SharePoint 2010 Site, List, and List Items Using the Client Object Model

▶ Task 1: Create a console application

1. On the **Start** menu, click **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.

   Microsoft® Visual Studio® 2010 opens.

2. On the **File** menu, point to **New**, and then click **Project**.
3. In the **Installed Templates** section, click the **Visual C#** node, and then click **Console Application**.
4. In the **.NET Framework version** drop-down list, click **.Net Framework 3.5**.
5. In the **Name** text box, type **Lab08**.
6. In the **Location** text box, type **E:\Student\Lab08\Starter**, and then click **OK**.

   The project is created.

▶ Task 2: Add a reference to the Microsoft SharePoint Client Object Model DLLs

1. In the Solutions Explorer window, right-click the **References** node, and click **Add Reference**.
2. In the **Add Reference** dialog box, click the **Browse** tab, and then browse to the following location: **C:\Program Files\Common Files\Microsoft Shared \Web Server Extensions\14\ISAPI**
3. Click **Microsoft.SharePoint.Client.dll**, hold down the CTRL key, click **Microsoft.SharePoint.Client.Runtime.dll**, and then click **OK**.

   The references to the Microsoft SharePoint® Client Object Model DLLs are added to the Solution Explorer window under References.

▶ Task 3: Add a using reference for the SharePoint Client Object Model

- At the top of the code file, after the last using statement, add the following code:

```
using Microsoft.SharePoint.Client;
```

▶ Task 4: Add code to the Main() method

- In the code window, add the following code between the braces for the **Main** method:

```
string option = string.Empty;
Console.Write("Options: [C] Create Skills Matrix: | [V] View Skills Matrix: ");
option = Console.ReadLine().ToUpper();
while ((option != "C") && (option != "V"))
{
  Console.Write(
    "Options: [C] Create Matrix: | [V] View Matrix: ");
```

```
  option = Console.ReadLine().ToUpper();
}
if (option == "C")
{
  createSkillsMatrix();
  return;
}
```

▶ Task 5: Create the createSkillsMatrix() Method

- In the code window, after the closing brace for the **Main()** method, add the following code:

```
static void createSkillsMatrix()
{
  ClientContext remoteCtx = null;
  try
  {
    Console.Write(
      "Enter the URL of a SharePoint site to house the Skills Matrix"
      + " (e.g. http://myserver): ");
    string siteUrl = Console.ReadLine().ToLower();
    while (!siteUrl.StartsWith("http://"))
    {
      Console.Write(
       "Enter the URL of a SharePoint site to house the Skills Matrix"
       + " (e.g. http://myserver): ");
      siteUrl = Console.ReadLine().ToLower();
    }
    remoteCtx = new ClientContext(siteUrl);
    Web remoteWeb = remoteCtx.Web;
    remoteCtx.Load(remoteWeb);
    WebCreationInformation skillsInfo = new WebCreationInformation();
    skillsInfo.Title = "Skills Matrix";
    skillsInfo.Language = 1033;
    skillsInfo.WebTemplate = "STS#1";
    skillsInfo.Url = "Skills";
    skillsInfo.UseSamePermissionsAsParentSite = true;
    Web skillsWeb = remoteWeb.Webs.Add(skillsInfo);
    skillsWeb.QuickLaunchEnabled = true;
    Console.Write("Skills Site is being created...");
    remoteCtx.ExecuteQuery();
    Console.WriteLine(" Success!");
    Console.Write("Skills Matrix is being created...");
    ListCreationInformation jobDefInfo =
      new ListCreationInformation();
    jobDefInfo.TemplateType = (int)ListTemplateType.GenericList;
    jobDefInfo.Title = "Jobs";
    jobDefInfo.QuickLaunchOption = QuickLaunchOptions.On;
    List jobDef = skillsWeb.Lists.Add(jobDefInfo);
    jobDef.Fields.AddFieldAsXml(
      @"<Field Type='Text' DisplayName='Description'/>",true,
      AddFieldOptions.DefaultValue);
    jobDef.Fields.AddFieldAsXml(
      @"<Field Type='Choice' DisplayName='Level' Format='Dropdown'>"
      + "<Default>Junior</Default>"
      + "<CHOICES>"
      + "<CHOICE>N/A</CHOICE>"
      + "<CHOICE>Junior</CHOICE>"
      + "<CHOICE>Senior</CHOICE>"
      + "<CHOICE>Lead</CHOICE>"
      + "</CHOICES>"
      + "</Field>", true, AddFieldOptions.DefaultValue);
    ListItemCreationInformation jobInfo =
```

```
    new ListItemCreationInformation();
ListItem job = jobDef.AddItem(jobInfo);
job["Title"] = "Developer";
job["Description"] = "Writes Code to Spec";
job["Level"] = "Junior";
job.Update();
job = jobDef.AddItem(jobInfo);
job["Title"] = "Developer";
job["Description"] = "Creates Specs and Writes Code";
job["Level"] = "Senior";
job.Update();
job = jobDef.AddItem(jobInfo);
job["Title"] = "Developer";
job["Description"] = "Manages Development Teams";
job["Level"] = "Lead";
job.Update();
job = jobDef.AddItem(jobInfo);
job["Title"] = "Analyst";
job["Description"] = "Defines business processes";
job["Level"] = "Junior";
job.Update();
job = jobDef.AddItem(jobInfo);
job["Title"] = "Analyst";
job["Description"] =
  "Defines business processes and creates reports";
job["Level"] = "Senior";
job.Update();
job = jobDef.AddItem(jobInfo);
job["Title"] = "Analyst";
job["Description"] = "Manages Analysis Teams";
job["Level"] = "Lead";
job.Update();
ListCreationInformation skillDefInfo =
  new ListCreationInformation();
skillDefInfo.TemplateType = (int)ListTemplateType.GenericList;
skillDefInfo.Title = "Skills";
skillDefInfo.QuickLaunchOption = QuickLaunchOptions.On;
List skillDef = skillsWeb.Lists.Add(skillDefInfo);
skillDef.Fields.AddFieldAsXml(
  @"<Field Type='Text' DisplayName='Description'/>",
  true, AddFieldOptions.DefaultValue);
skillDef.Fields.AddFieldAsXml(
  @"<Field Type='Choice' DisplayName='Importance'"
  + " Format='Dropdown'>"
  + "<Default>Essential</Default>"
  + "<CHOICES>"
  + "<CHOICE>Essential</CHOICE>"
  + "<CHOICE>Preferred</CHOICE>"
  + "<CHOICE>Optional</CHOICE>"
  + "</CHOICES>"
  + "</Field>", true, AddFieldOptions.DefaultValue);
ListItemCreationInformation skillInfo =
  new ListItemCreationInformation();
ListItem skill = skillDef.AddItem(skillInfo);
skill["Title"] = "Write Code";
skill["Importance"] = "Essential";
skill.Update();
skillInfo = new ListItemCreationInformation();
skill = skillDef.AddItem(skillInfo);
skill["Title"] = "Write Code";
skill["Importance"] = "Preferred";
skill.Update();
skillInfo = new ListItemCreationInformation();
skill = skillDef.AddItem(skillInfo);
```

```
skill["Title"] = "Write Code";
skill["Importance"] = "Optional";
skill.Update();
skill = skillDef.AddItem(skillInfo);
skill["Title"] = "Make Technical Decisions";
skill["Importance"] = "Essential";
skill.Update();
skillInfo = new ListItemCreationInformation();
skill = skillDef.AddItem(skillInfo);
skill["Title"] = "Make Technical Decisions";
skill["Importance"] = "Preferred";
skill.Update();
skillInfo = new ListItemCreationInformation();
skill = skillDef.AddItem(skillInfo);
skill["Title"] = "Make Technical Decisions";
skill["Importance"] = "Optional";
skill.Update();
skill = skillDef.AddItem(skillInfo);
skill["Title"] = "Analyze Business Processes";
skill["Importance"] = "Essential";
skill.Update();
skillInfo = new ListItemCreationInformation();
skill = skillDef.AddItem(skillInfo);
skill["Title"] = "Analyze Business Processes";
skill["Importance"] = "Preferred";
skill.Update();
skillInfo = new ListItemCreationInformation();
skill = skillDef.AddItem(skillInfo);
skill["Title"] = "Analyze Business Processes";
skill["Importance"] = "Optional";
skill.Update();
skill = skillDef.AddItem(skillInfo);
skill["Title"] = "Write Reports";
skill["Importance"] = "Essential";
skill.Update();
skillInfo = new ListItemCreationInformation();
skill = skillDef.AddItem(skillInfo);
skill["Title"] = "Write Reports";
skill["Importance"] = "Preferred";
skill.Update();
skillInfo = new ListItemCreationInformation();
skill = skillDef.AddItem(skillInfo);
skill["Title"] = "Write Reports";
skill["Importance"] = "Optional";
skill.Update();
remoteCtx.ExecuteQuery();
remoteCtx.Load(jobDef);
remoteCtx.Load(skillDef);
remoteCtx.ExecuteQuery();
ListCreationInformation mixerInfo = new ListCreationInformation();
mixerInfo.TemplateType = (int)ListTemplateType.GenericList;
mixerInfo.Title = "Mashup";
mixerInfo.QuickLaunchOption = QuickLaunchOptions.Off;
List mixer = skillsWeb.Lists.Add(mixerInfo);
mixer.Fields.AddFieldAsXml(@"<Field Type='Integer' "
+ " Name='LookupJobs' DisplayName='LookupJobs' Required='TRUE'/>",
  true, AddFieldOptions.DefaultValue);
mixer.Fields.AddFieldAsXml(
  @"<Field Type='Integer' Name='LookupSkills' "
  + "DisplayName='LookupSkills' Required='TRUE'/>",
  true, AddFieldOptions.DefaultValue);
Field titleField = mixer.Fields.GetByInternalNameOrTitle("Title");
titleField.Hidden = true;
titleField.Update();
```

```
    mixer.Hidden = true;
    mixer.Update();
    remoteCtx.ExecuteQuery();
    Console.WriteLine(" Success!");
    Console.Write("Press any key to exit...");
    Console.ReadLine();
  }
  catch(Exception ex)
  {
    Console.WriteLine();
    Console.WriteLine(ex.Message);
    Console.Write("Press any key to exit...");
    Console.ReadLine();
  }
  finally
  {
    remoteCtx.Dispose();
  }
}
```

Exercise 2: Building and Using the Console Application

▶ Task 1: Build the console application

1. In the Solution Explorer window, right-click the **Lab08** project, and click **Build**.

   Verify the Build Succeeded message appears below the code pane.

2. On the **Debug menu**, click **Start Without Debugging**.
3. In the console window, review the text and press C.
4. At the **Enter the URL of a SharePoint site to house the Skills Matrix** prompt, type **http://SharePoint**, and then press ENTER.
5. When you are notified that the Skills Site and the Skills Matrix have been created, press ENTER twice to close the console application.

▶ Task 2: Review the Skills Matrix site

1. Open Windows® Internet Explorer® and browse to **http://sharepoint**.
2. On the **Quick Launch** menu, click **All Site Content**.
3. In the **Sites and Workspaces** section, click **Skills Matrix**.
4. On the **Quick Launch** menu, click **Skills**.

   Review the skills that were created in the console application.

5. On the **Quick Launch** menu, click **Jobs**.

   Review the Jobs that were created in the console application.

Module 9

Lab Answer Key: Developing Interactive User Interfaces

Contents:

Lab: Developing User Interface Components for SharePoint 2010 Solutions

Log On to the Virtual Machine for This Lab

For this lab, you use the available virtual machine environment. Before you begin the lab, log on to the **10175-LON-DEV-09** virtual machine as **Administrator**, with the password **P@ssw0rd**.

Exercise 1: Creating a Site Actions Menu Item

▶ Task 1: Create an empty SharePoint Project

1. On the **Start** menu, click **All Programs**, click **Microsoft Visual Studio 2010** and then click **Microsoft Visual Studio 2010**.
2. On the **File** menu, point to **New**, and then click **Project**.
3. In the **Installed Templates** section, expand the **Visual C#** node, expand the **SharePoint** node and then click **2010**.
4. Click **Empty SharePoint Project**.
5. In the **Name** text box, type **Lab09**.
6. In the **Location** text box, type **E:\Student\Lab09\Starter**.
7. Click **OK**.

   The Microsoft® SharePoint® Customization Wizard appears.
8. In the **What local site do you want to use for debugging?** text box, type **http://sharepoint**.
9. In the **What is the trust level for this SharePoint** solution section, click the **Deploy as a farm solution** option.
10. Click **Finish**.

    The project is created.

▶ Task 2: Add an empty element to the project

1. In the Solution Explorer window, right-click **Lab09** and click **Add**, and then click **New Item**.
2. In the **Add New Item** dialog, in the list of installed templates, click **Empty Element**.
3. In the **Name** textbox, type **SiteActions_Features** and click **Add**.

   The SiteActions_Features node is added to the project and the Elements.xml file opens in the code window.

▶ Task 3: Edit the Elements.xml file to add a new menu option to the Site Actions menu

- Replace the elements.xml file with the following markup:

```
<?xml version="1.0" encoding="utf-8"?>
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
  <CustomAction Id="NewUIActionsMenu"
    GroupId="SiteActions"
      Location="Microsoft.SharePoint.StandardMenu"
      Sequence="1970"
      Title="Manage Features">
    <UrlAction Url="/_layouts/ManageFeatures.aspx" />
  </CustomAction>
```

```
</Elements>
```

Task 4: Build and test the item

1. In the Solution Explorer window, right-click the **Lab09** project and click **Deploy**.
2. Start Windows® Internet Explorer® and browse to the site **http://sharepoint**.
3. On the **Site Actions** menu, click **Manage Features**.

   The new option called Manage Features appears on the Site Actions menu and navigates the user to /_layouts/ManageFeatures.aspx.
4. On the **ManageFeatures.aspx** page, locate the **Lab09 Feature1** item and click **Deactivate**.
5. On the **You are about to deactivate the Lab09 Feature1 feature** warning message, click **Deactivate this feature**.
6. Click the **Site Actions** menu and review the menu items.

   Note that the Manage Features option no longer appears on this list.
7. On the **ManageFeatures.aspx** page, locate the **Lab09 Feature1** item and click **Activate**.

   The Manage Features option is re-enabled on the menu.

Exercise 2: Creating a Ribbon Item

▶ Task 1: Add an empty element to the project

1. Switch back to the Visual Studio 2010 Lab09 project.
2. In the Solution Explorer window, right-click **Lab09**, click **Add** and then click **New Item**.
3. Click **Empty Element**.
4. In the **Name** textbox, type **HelpRibbon** and click **Add**.

   The HelpRibbon node is added to the project and the Elements.xml file opens in the code window.

▶ Task 2: Edit the Elements.xml file to add a new item to the Documents ribbon

1. Replace the Elements.xml file with the following markup:

```
<?xml version="1.0" encoding="utf-8"?>
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
  <CustomAction
  Id="NewUIRibbonControl"
  RegistrationType="List"
  RegistrationId="101"
  Location="CommandUI.Ribbon">
  <CommandUIExtension>
    <CommandUIDefinitions>
      <CommandUIDefinition
        Location="Ribbon.Documents.New.Controls._children">
        <Button
          Id="NewUIRibbonControl.ShowHelp"
          Alt="Help"
          Sequence="1981"
          Command="ShowHelp"
          Image32by32="/_layouts/images/lab09/dochelp.png"
          LabelText="Help Videos"
          TemplateAlias="o1"/>
        </CommandUIDefinition>
      </CommandUIDefinitions>
      <CommandUIHandlers>
        <CommandUIHandler
          Command="ShowHelp"
          CommandAction="javascript:window.open(
'http://msdn.microsoft.com/en-gb/sharepoint/ee663870.aspx');" />
      </CommandUIHandlers>
    </CommandUIExtension>
  </CustomAction>
</Elements>
```

2. On the **File** menu, click **Save All**.

▶ Task 3: Add images to the SharePoint project

1. In the Solution Explorer window, right-click **Lab09**, click **Add** and then click **SharePoint Images Mapped Folder**.
2. In the Solution Explorer window, under the **Images** node, right-click **Lab09** and click **Add** and then click **Existing Item**.
3. Locate the file **E:\Student\Lab09\docHelp.png** and click **Add**.

▶ Task 4: Build and test the ribbon item

1. In the Solution Explorer window, right-click the **Lab09** project and click **Deploy**.

2. Switch back to **Internet Explorer** and browse to the site **http://sharepoint**.
3. On the **Quick Launch** bar, click **Shared Documents**.
4. On the ribbon, click the **Documents** tab.
5. On the ribbon, in the **New** section, click **Help Videos**.

   This is the ribbon item that you have just added.
6. Close all instances of Internet Explorer.

Exercise 3: Creating a Client-Side Dialog

Task 1: Add a Visual Web Part to the project

1. Switch back to the Visual Studio 2010 Lab09 project.
2. In the Solution Explorer window, right-click **Lab09** and click **Add** and then click **New Item**.
3. In the **Add New Item** dialog, in the list of installed templates, click **Visual Web Part**.
4. In the **Name** textbox, type **SkillsManager** and click **Add**.

   The Visual Web Part is created and added to the project.

Task 2: Add a TreeView control to the Visual Web Part

1. Add the following markup to the end of the SkillsManagerUserControl.ascx file in the Visual Web Part:

```
<asp:TreeView ID="skillsMap" runat="server" ShowLines="true">
</asp:TreeView>
```

2. On the **View** menu, click **Code**.

Task 3: Add code for the TreeView control

1. Locate the **using** statements at the top of the code file and add a using statement for **Microsoft.SharePoint**. Your using statements should look as follows:

```
using System;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using Microsoft.SharePoint;
```

2. In the **Page_Load** method, add a **try/catch** block. Your code should look as follows:

```
namespace Lab09.SkillsManager
{
  public partial class SkillsManagerUserControl : UserControl
  {
    protected void Page_Load(object sender, EventArgs e)
    {
      try
      {

      }
      catch (Exception ex)
      {

      }
    }
  }
}
```

3. In the **try** block, add the following code:

```
skillsMap.Nodes.Clear();
SPWeb thisWeb = SPContext.Current.Web;
SPWeb skillsWeb = thisWeb.Webs["Skills"];
SPList jobs = skillsWeb.Lists["Jobs"];
SPList mashup = skillsWeb.Lists["Mashup"];
SPList skills = skillsWeb.Lists["Skills"];
foreach (SPListItem item in jobs.Items)
```

```
{
  TreeNode job = new TreeNode(item["Title"].ToString() + ": "
    + item["Level"].ToString());
  int jobID = item.ID;
  SPQuery skillsMapping = new SPQuery();
  skillsMapping.ViewFields = "<FieldRef Name='LookupSkills' />";
  skillsMapping.Query = "<Where><Eq><FieldRef Name='LookupJobs' />"
    + "<Value Type='Integer'>" + jobID.ToString()
    + "</Value></Eq></Where>";
  SPListItemCollection mappings = mashup.GetItems(skillsMapping);
  if (mappings.Count > 0)
  {
    SPQuery assignedSkills = new SPQuery();
    assignedSkills.ViewFields = "<FieldRef Name='ID' />"
      + "<FieldRef Name='Title' /><FieldRef Name='Importance' />";
    SPListItemCollection jobSkills = skills.GetItems(assignedSkills);
    foreach (SPListItem jobSkill in jobSkills)
    {
      foreach (SPListItem mapping in mappings)
      {
        if(mapping["LookupSkills"].ToString()==jobSkill.ID.ToString())
        {
          job.ChildNodes.Add(new TreeNode(jobSkill["Title"].ToString()
            + ": " + jobSkill["Importance"].ToString()));
        }
      }
    }
  }
  skillsMap.Nodes.Add(job);
}
skillsMap.ExpandAll();
```

4. In the **catch** block, add the following code:

```
catch (Exception ex)
{
  skillsMap.Nodes.Add(new TreeNode(ex.Message));
}
```

▶ Task 4: Add HTML markup to the SkillManager Visual Web Part

1. In the Solution Explorer window, expand the **SkillsManager** node, and double-click **SkillsManagerUserControl.ascx**.
2. Click the **Source** tab.
3. After the closing tag for the **TreeView** element, add the following markup:

```
<br/><br/>
```

4. At the end of the existing markup, add the following HTML link code:

```
<a href="javascript:showSkillAdder();" id="linkAdder"
  style="display:none;">
  &raquo;Add Skill Mapping
</a>
```

5. At the end of the existing markup, add the following containing a table:

```
<div id="divSkillsManager" style="display:none; padding:5px">
  <table cellpadding='3' cellspacing='3'>
    <tr>
```

```
      <td align='center' colspan='2'><b>Skill Mapper</b><br/></td>
    </tr>
    <tr>
      <td>Select a Job:</td>
      <td><select id="Jobs" name="Jobs" /></td>
    </tr>
    <tr>
      <td>Select a Skill:</td>
      <td><select id="Skills" name="Skills" /></td>
    </tr>
    <tr>
      <td colspan='2' align='right'>
        <input type="button" value="Add Skill Mapping"
          style='width:125px;' onclick="addSkillMapping()" />
        <br />
        <input type="button" value="Cancel" style='width:125px'
          onclick="onCancel()" />
      </td>
    </tr>
  </table>
</div>
```

6. At the end of the existing markup, add the following SharePoint script link:

```
<SharePoint:ScriptLink ID="showDialog" runat="server" Name="sp.js"
Localizable="false" LoadAfterUI="true" />
```

7. At the end of the existing markup, add the following JavaScript code:

```
<script language="ecmascript" type="text/ecmascript">

</script>
```

8. Between the begin and end Script tags, add the following variables:

```
var thisDialog;
var jobSelector;
var skillSelector;
var clientCtx;
var skillsWeb;
var mashupList;
var jobList;
var skillList;
var jobItems;
var skillItems;
var mashupItems;
var camlQuery;
var listItemEnumerator;
var listItem;
var selectedJob;
var selectedSkill;
```

9. After the JavaScript variable definition, and before the closing </script> tag, add the following initialization code:

```
_spBodyOnLoadFunctionNames.push("Initialize()");
```

10. After the Initialization code, and before the closing </script> tag, add the following **Initialize()** function:

```
function Initialize() {
```

```
  clientCtx = new SP.ClientContext('/skills');
  skillsWeb = clientCtx.get_web();
  jobList = skillsWeb.get_lists().getByTitle("Jobs");
  skillList = skillsWeb.get_lists().getByTitle("Skills");
  mashupList = skillsWeb.get_lists().getByTitle("Mashup");
  camlQuery = new SP.CamlQuery();
  camlQuery.set_viewXml('<View><RowLimit>50</RowLimit></View>');
  jobItems = jobList.getItems(camlQuery);
  skillItems = skillList.getItems(camlQuery);
  clientCtx.load(skillsWeb);
  clientCtx.load(jobList);
  clientCtx.load(skillList);
  clientCtx.load(jobItems, 'Include(Id, Title, Level)');
  clientCtx.load(skillItems, 'Include(Id, Title, Importance)');
  clientCtx.executeQueryAsync(onSkillsWebLoaded,
  errOnSkillsWebLoaded);
}
```

11. After the Initialize() function, and before the closing </script> tag, add the following **errOnSkillsWebLoaded()** function:

```
function errOnSkillsWebLoaded(sender, args) {
  alert(args.get_message());
}
```

12. After the errOnSkillsWebLoaded() function, and before the closing </script> tag, add the following **onSkillsWebLoaded()** function:

```
function onSkillsWebLoaded() {
  var linkAdder = document.getElementById("linkAdder");
  linkAdder.style.display = "inline";
}
```

13. After the onSkillsWebLoaded() function, and before the closing </script> tag, add the following **showSkillAdder()** function:

```
function showSkillAdder() {
  jobSelector = document.getElementById("Jobs");
  skillSelector = document.getElementById("Skills");
  jobSelector.options.length = 0;
  skillSelector.options.length = 0;
  listItemEnumerator = jobItems.getEnumerator();
  while (listItemEnumerator.moveNext()) {
    listItem = listItemEnumerator.get_current();
    jobSelector.options[jobSelector.options.length]
      = new Option(listItem.get_item('Title') + ': '
      + listItem.get_item('Level'), listItem.get_id());
  }
  listItemEnumerator = skillItems.getEnumerator();
  while (listItemEnumerator.moveNext()) {
    listItem = listItemEnumerator.get_current();
    skillSelector.options[skillSelector.options.length]
      = new Option(listItem.get_item('Title') + ': '
      + listItem.get_item('Importance'), listItem.get_id());
  }
  var divSkillsManager = document.getElementById("divSkillsManager");
  divSkillsManager.style.display = "block";
  var dialog = { html: divSkillsManager,
    title: 'Add Skills Mapping', allowMaximize: true,
    showClose: false, width: 400, height: 200};
  thisDialog = SP.UI.ModalDialog.showModalDialog(dialog);
```

```
}
```

14. After the showSkillAdder() function, and before the closing </script> tag, add the following **hideSkillMapper()** function:

```
function hideSkillMapper() {
  thisDialog.close();
  window.location.reload(true);
}
```

15. After the hideSkillMapper() function, and before the closing </script> tag, add the following **addSkillMapping()** function:

```
function addSkillMapping() {
  selectedJob =
    jobSelector.options[jobSelector.selectedIndex].value;
  selectedSkill =
    skillSelector.options[skillSelector.selectedIndex].value;
  clientCtx.load(mashupList);
  camlQuery = new SP.CamlQuery();
  camlQuery.set_viewXml('<View><Query><Where><Eq>' +
    '<FieldRef Name=\'LookupJobs\'/><Value Type=\'Integer\'>'
    + selectedJob + '</Value>'
    + '</Eq></Where></Query><RowLimit>50</RowLimit></View>');
  mashupItems = mashupList.getItems(camlQuery);
  clientCtx.load(mashupItems);
  clientCtx.executeQueryAsync(onCurrentMapLoaded,
    errOnCurrentMapLoaded);
}
```

16. After the addSkillMapping() function, and before the closing </script> tag, add the following **onCurrentMapLoaded()** function:

```
function onCurrentMapLoaded() {
  var alreadySelected = false;
  listItemEnumerator = mashupItems.getEnumerator();
  while (listItemEnumerator.moveNext()) {
    listItem = listItemEnumerator.get_current();
    if (listItem.get_item('LookupSkills') == selectedSkill) {
      alreadySelected = true;
      break;
    }
  }
  if (!alreadySelected) {
    var listItemInfo = new SP.ListItemCreationInformation();
    var newListItem = mashupList.addItem(listItemInfo);
    newListItem.set_item('LookupJobs', selectedJob);
    newListItem.set_item('LookupSkills', selectedSkill);
    newListItem.update();
    clientCtx.executeQueryAsync(onMappingAdded, errOnMappingAdded);
    return;
  }
  hideSkillMapper();
}
```

17. After the onCurrentMapLoaded() function, and before the closing </script> tag, add the following **errOnCurrentMapLoaded()** function:

```
function errOnCurrentMapLoaded(sender, args) {
  alert(args.get_message());
```

```
}
```

18. After the errOnCurrentMapLoaded() function, and before the closing </script> tag, add the following **onMappingAdded()** function:

```
function onMappingAdded(sender, args) {
  hideSkillMapper();
}
```

19. After the onMappingAdded() function, and before the closing </script> tag, add the following **errOnMappingAdded()** function:

```
function errOnMappingAdded(sender, args) {
  alert(args.get_message());
}
```

20. After the errOnMappingAdded() function, and before the closing </script> tag, add the following **onSkillMapped()** function:

```
function onSkillMapped() {
  hideSkillMapper();
}
```

21. After the onSkillMapped() function, and before the closing </script> tag, add the following **onCancel()** function:

```
function onCancel() {
  hideSkillMapper();
}
```

22. On the **File** menu, click **Save All**.

▶ Task 5: Deploy and test the client-side dialog

1. In the Solution Explorer window, right-click the **Lab09** project and click **Deploy**.
2. Start Internet Explorer and browse to **http://sharepoint**.
3. On the **Site Actions** menu, click **New Page**.
4. In the **New page name** text box, type **Skills** and then click **Create**.
5. On the ribbon, click the **Insert** tab.
6. On the ribbon, click **Web Part**.
7. In the **Categories** section, click **Custom**.
8. In the **Web Parts** section, click **SkillsManager**.
9. Click **Add**.

   The Web Part is added to the page.

10. On the ribbon, click **Save and Close**.

    Note the new Add Skill Mapping HTML link that appears in the Visual Web Part.

11. Click **Add Skill Mapping**.
12. In the dialog, in the **Select a job** drop-down list, click **Developer: Senior**.
13. In the **Select a Skill** drop-down list, click **Make Technical Decisions: Preferred**.
14. Click **Add Skill Mapping**.

Module 10

Lab Answer Key: Developing Silverlight Applications for SharePoint

Contents:

Lab: Developing Silverlight Applications by Using the SharePoint Client Object Model

Log On to the Virtual Machine for This Lab

For this lab, you use the available virtual machine environment. Before you begin the lab, you must log on to the **10175-LON-DEV-10** virtual machine as **Administrator** using the password **P@ssw0rd**.

Exercise 1: Creating a Silverlight Application

▶ Task 1: Create a Silverlight project

1. On the **Start** menu, click **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.

   Microsoft® Visual Studio® opens.

2. On the **File** menu, point to **New**, and then click **Project**.
3. In the **Installed Templates** section, expand the **Visual C#** node, and then click **Silverlight**.
4. Click **Silverlight Application**.
5. In the **Name** text box, type **Lab10**.
6. In the **Location** text box, type **E:\Student\Lab10\Starter**, and then click **OK**.
7. In the **New Silverlight Application** dialog box, clear the **Host the Silverlight application in a new Web site** check box, and then click **OK**.

   The Microsoft Silverlight® project is created.

▶ Task 2: Prepare the Silverlight user interface

1. In the **UserControl** element, after the **d:DesignWidth="400"** attribute and before the closing Extensible Markup Language (XML) tag, add the following attributes:

```
Height="430" Width="600"
```

2. In the **MainPage.xaml** file, in the **xaml** view, add the following code between the open and close **<Grid>** tags:

```
<ProgressBar Height="20" HorizontalAlignment="Left"
  Background="Black" Margin="10,10,10,0" Name="Loader"
  VerticalAlignment="Top" Width="580" />
<TextBlock Height="20" HorizontalAlignment="Left"
  Margin="10,35,0,0" Name="Status" Text="Status"
  VerticalAlignment="Top"/>
<ComboBox Name="LibraryPicker" Height="30" Width="500"
  HorizontalAlignment="Left" Margin="10,60,0,0"
  VerticalAlignment="Top"/>
<StackPanel Orientation="Horizontal" Margin="10,100,0,0"
  Height="320">
<ScrollViewer Width="235" Height="310"  HorizontalAlignment="Left"
  VerticalAlignment="Top" HorizontalContentAlignment="Left"
  VerticalContentAlignment="Top">
  <StackPanel  Width="210"
    ScrollViewer.VerticalScrollBarVisibility="Auto"
    ScrollViewer.HorizontalScrollBarVisibility="Auto"
    Name="myContainer" HorizontalAlignment="Left"
    VerticalAlignment="Top"  />
```

```
</ScrollViewer>
<StackPanel Name="myMedia" Height="310" Width="340"
  Margin="10,0,0,0" HorizontalAlignment="Left"
  VerticalAlignment="Top"  />
</StackPanel>
```

3. On the **File** menu, click **Save All**.

Exercise 2: Developing the Silverlight Application

Task 1: Set the build location

1. In the Solution Explorer window, right-click **Lab10**, and then click **Properties**.
2. In the **Tabs** section, on the left side of the **Properties** page, click **Build**.
3. In the **Output path** section, click **Browse**, and then browse to the following location: **C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\14\TEMPLATE\LAYOUTS\ClientBin**
4. In the Select Output Path window, click **Select Folder**.
5. On the **File** menu, click **Save All**.
6. Close the **Properties** tab.

Task 2: Add a reference to the Microsoft SharePoint® client object model DLLs

1. In the Solution Explorer window, right-click the **References** node, and click **Add Reference**.
2. In the **Add Reference** dialog box, click the **Browse** tab, and then browse to the following location: **C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\14\Template\Layouts\ClientBin**
3. Click **Microsoft.SharePoint.Client.Silverlight.dll**, hold down the CTRL key, click **Microsoft.SharePoint.Client.Silverlight.Runtime.dll**, and then click **OK**.

Task 3: Add using statements to App.xaml.cs

1. In the Solution Explorer window, expand the **App.xaml** node, and double-click the **App.xaml.cs** file.
2. At the end of the current **using** statements, add the following code:

```
using Microsoft.SharePoint.Client;
using System.Threading;
```

Task 4: Add the ApplicationContext to the Application_Startup method

1. In the **Application_Startup** method, before the existing line of code, add the following code:

```
ApplicationContext.Init(
  e.InitParams, SynchronizationContext.Current);
```

2. On the **File** menu, click **Save All**.
3. Close the **App.xaml.cs** file.

Task 5: Add using statements to MainPage.xaml.cs

1. In the Solution Explorer window, expand the **MainPage.xaml** node, and double-click the **MainPage.xaml.cs** file.
2. At the end of the current **using** statements, add the following code:

```
using Microsoft.SharePoint.Client;
using System.Threading;
using System.Windows.Media.Imaging;
```

Task 6: Add variables to MainPage.xaml.cs

1. In the **MainPage.xaml.cs** file, in the class **MainPage**, add the following code:

```
ClientContext clientCtx;
Microsoft.SharePoint.Client.List mediaLib;
int fileTracker = 0;
int rowTracker = 0;
StackPanel row = new StackPanel();
```

2. On the **File** menu, click **Save All**.

Task 7: Add the Loaded event to MainPage

1. In the Solution Explorer window, open the **MainPage.xaml**.
2. In the **<Grid>** element, after the background element and before the closing XML tag, add the following code:

```
Loaded="LayoutRoot_Loaded"
```

3. On the **File** menu, click **Save All**.
4. On the **View** menu, click **Code**.

Task 8: Add event methods to MainPage.xaml.cs

1. After the **MainPage()** method, add the following code:

```
private void LayoutRoot_Loaded(object sender, RoutedEventArgs e)
{
  Loader.Maximum = 2;
  Loader.Value = 0;
  Status.Text = "Connecting to Web...";
  clientCtx = new ClientContext(ApplicationContext.Current.Url);
  clientCtx.Load(clientCtx.Web);
  clientCtx.ExecuteQueryAsync(updateConnectionStatus,
    errUpdateConnectionStatus);
}
```

2. After the **LayoutRoot_Loaded()** method, add the following **updateConnectionStatus()** method:

```
void updateConnectionStatus(Object sender,
  ClientRequestSucceededEventArgs e)
{
  Dispatcher.BeginInvoke(makeProgressWebConnection);
}
```

3. After the **updateConnectionStatus()** method, add the following **errUpdateConnectionStatus()** method:

```
void errUpdateConnectionStatus(Object sender, ClientRequestFailedEventArgs e)
{
  Status.Text = e.Message;
}
```

4. After the **errUpdateConnectionStatus()** method, add the following **makeProgressWebConnection()** method:

```
void makeProgressWebConnection()
{
  Loader.Value++;
  Status.Text = "Web Connected. Connecting to media stores...";
  clientCtx.Load(clientCtx.Web.Lists, lists => lists
    .Include(list => list.Title, list => list.Id)
```

```
    .Where(list => list.BaseType==BaseType.DocumentLibrary
      && !list.Hidden));
  clientCtx.ExecuteQueryAsync(updateLists, errUpdateLists);
}
```

5. After the **makeProgressWebConnection()** method, add the following **updateLists()** method:

```
void updateLists(Object sender, ClientRequestSucceededEventArgs e)
{
  Dispatcher.BeginInvoke(makeProgressListConnection);
}
```

6. After the **updateLists()** method, add the following **errUpdateLists()** method:

```
void errUpdateLists(Object sender, ClientRequestFailedEventArgs e)
{
  Status.Text = e.Message;
}
```

7. After the **errUpdateLists ()** method, add the following **makeProgressListConnection()** method:

```
void makeProgressListConnection()
{
  Loader.Value++;
  Status.Text = "Now showing all media stores";
  foreach (List mediaStore in clientCtx.Web.Lists)
  {
    ListBoxItem mediaLibPicker = new ListBoxItem();
    mediaLibPicker.Content = mediaStore.Title;
    mediaLibPicker.Tag = mediaStore.Id;
    mediaLibPicker.MouseLeftButtonUp += new
      MouseButtonEventHandler(mediaLibPicker_MouseLeftButtonUp);
    LibraryPicker.Items.Add(mediaLibPicker);
  }
}
```

8. After the **makeProgressListConnection()** method, add the following **mediaLibPicker_MouseLeftButtonUp()** method:

```
void mediaLibPicker_MouseLeftButtonUp(object sender, MouseButtonEventArgs e)
{
  ListBoxItem src = (ListBoxItem)sender;
  string libID = src.Tag.ToString();
  mediaLib = clientCtx.Web.Lists.GetById(new Guid(libID));
  clientCtx.Load(mediaLib);
  clientCtx.Load(mediaLib.RootFolder);
  clientCtx.Load(mediaLib.RootFolder.Files);
  clientCtx.ExecuteQueryAsync(getListData, errGetListData);
}
```

9. After the **mediaLibPicker_MouseLeftButtonUp()** method, add the following **getListData()** method:

```
void getListData(Object sender, ClientRequestSucceededEventArgs e)
{
  Dispatcher.BeginInvoke(loadFiles);
}
```

10. After the **getListData()** method, add the following **errGetListData()** method:

```
void errGetListData(Object sender, ClientRequestFailedEventArgs e)
{
```

```
  Status.Text = e.Message;
}
```

11. After the **errGetListData()** method, add the following **loadFiles ()** method:

```
private void loadFiles()
{
  myContainer.Children.Clear();
  Loader.Maximum = mediaLib.RootFolder.Files.Count;
  if (Loader.Maximum != 0)
  {
    Loader.Value = 0;
    Status.Text = "Loading Files...";
    fileTracker = 0;
    foreach (File fle in mediaLib.RootFolder.Files)
    {
      clientCtx.Load(fle);
      clientCtx.ExecuteQueryAsync(addFileToUI, errAddFileToUI);
    }
    return;
  }
  Status.Text = "There are no files to show from this library";
}
```

12. After the **loadFiles()** method, add the following **addFileToUI()** method:

```
void addFileToUI(Object sender, ClientRequestSucceededEventArgs e)
{
  Dispatcher.BeginInvoke(addFile);
}
```

13. After the **addFileToUI()** method, add the following **errAddFileToUI()** method:

```
void errAddFileToUI(Object sender, ClientRequestFailedEventArgs e)
{
  Status.Text = e.Message;
}
```

14. After the **errAddFileToUI()** method, add the following **addFile()** method:

```
void addFile()
{
  string fName = mediaLib.RootFolder.Files[fileTracker].Name;
  string fUrl =
    mediaLib.RootFolder.Files[fileTracker].ServerRelativeUrl;
  fUrl = clientCtx.Url + fUrl;
  if ((rowTracker % 2) == 0)
  {
    row = new StackPanel();
    row.Orientation = Orientation.Horizontal;
    myContainer.Children.Add(row);
  }
  if ((fName.ToLower().EndsWith(".png")) ||
    (fName.ToLower().EndsWith(".jpeg")) ||
    (fName.ToLower().EndsWith(".jpg")))
  {
    Image img = new Image();
    img.MaxWidth = 100;
    img.MaxHeight = 50;
    img.Stretch = Stretch.Uniform;
    BitmapImage bitMap = new BitmapImage(
      new Uri(fUrl, UriKind.Absolute));
```

```
    img.Source = bitMap;
    ContentControl mediaButton = new ContentControl();
    mediaButton.Margin = new Thickness(2.5);
    mediaButton.Tag = fUrl;
    ToolTipService.SetToolTip(mediaButton, fName);
    mediaButton.Cursor = Cursors.Hand;
    mediaButton.MouseLeftButtonUp += new
      MouseButtonEventHandler(mediaButton_MouseLeftButtonUp);
    mediaButton.Content = img;
    row.Children.Add(mediaButton);
    rowTracker++;
  }
  if (fName.ToLower().EndsWith(".wmv"))
  {
    MediaElement media = new MediaElement();
    media.MaxWidth = 100;
    media.MaxHeight = 50;
    media.Stretch = Stretch.Uniform;
    media.Source = new Uri(fUrl, UriKind.Absolute);
    media.AutoPlay = false;
    ContentControl mediaButton = new ContentControl();
    mediaButton.Margin = new Thickness(2.5);
    mediaButton.Tag = fUrl;
    ToolTipService.SetToolTip(mediaButton, fName);
    mediaButton.Cursor = Cursors.Hand;
    mediaButton.MouseLeftButtonUp += new
      MouseButtonEventHandler(mediaButton_MouseLeftButtonUp);
    mediaButton.Content = media;
    row.Children.Add(mediaButton);
    rowTracker++;
  }
  Loader.Value++;
  fileTracker++;
  if (fileTracker >= mediaLib.RootFolder.Files.Count)
  {
    Status.Text = "All files have now been loaded";
  }
}
```

15. After the **addFile()** method, add the following **mediaButton_MouseLeftButtonUp()** method:

```
void mediaButton_MouseLeftButtonUp(object sender, MouseButtonEventArgs e)
{
  myMedia.Children.Clear();
  ContentControl src = (ContentControl)sender;
  Uri fUri = new Uri(src.Tag.ToString(), UriKind.Absolute);
  if ((fUri.OriginalString.EndsWith(".png"))
    || (fUri.OriginalString.EndsWith(".jpg")))
  {
    Image img = new Image();
    img.MaxWidth = 340;
    img.MaxHeight = 310;
    img.Stretch = Stretch.Uniform;
    BitmapImage bitMap = new BitmapImage(fUri);
    img.Source = bitMap;
    myMedia.Children.Add(img);
    return;
  }
  if (fUri.OriginalString.EndsWith(".wmv"))
  {
    MediaElement media = new MediaElement();
    media.MaxWidth = 340;
    media.MaxHeight = 310;
```

```
    media.Stretch = Stretch.Uniform;
    media.Source = fUri;
    media.AutoPlay = true;
    myMedia.Children.Add(media);
  }
```

16. On the **File** menu, click **Save All**.

Task 9: Build and test the Silverlight application

1. In the Solution Explorer window, right-click **Lab10**, and click **Build**.
2. Switch back to Windows® Internet Explorer® and browse to the site **http://sharepoint**.
3. On the **Quick Launch** bar, click **Site Pages** and then click **Training**.
4. On the ribbon, click **Edit**.
5. On the ribbon, click the **Insert** tab.
6. On the ribbon, in the **Web Parts** section, click **Web Part**.
7. In the **Categories** section, in the **Media and Content** category, click **Silverlight Web Part**, and then click **Add**.
8. In the **Silverlight Web Part URL**, type **/_layouts/ClientBin/Lab10.xap**, and then click **OK**.
9. On the **Web Part** drop-down menu, click **Edit Web Part**.
10. On the right side of the screen, in the Silverlight Web Part properties panel, in the **Height** section, click the **Yes** option, and type **600** in the text box.
11. In the **Width** section, click **No, Adjust width to fit zone**, and then click **OK**.
12. On the ribbon, click **Save and Close**.
13. In the drop-down list in the Silverlight application, click **Shared Documents**.

    After a few seconds, video thumbnails are displayed.
14. Click each video thumbnail and watch a few seconds of each video.
15. Close all applications. You have now completed this lab.

Module 11

Lab Answer Key: Developing Sandboxed Solutions

Contents:

Lab: Creating Sandboxed Solutions for SharePoint 2010

Log On to the Virtual Machine for This Lab

For this lab, you use the available virtual machine environment. Before you begin the lab, you must log on to the **10175-LON-DEV-11** virtual machine as **Administrator**, using the password **P@ssw0rd**.

Exercise 1: Creating a Sandboxed Solution by Using Visual Studio 2010

▶ Task 1: Create an Empty SharePoint Project

1. On the **Start** menu, click **All Programs**, click **Microsoft Visual Studio 2010** and then click **Microsoft Visual Studio 2010**.
2. On the **File** menu, point to **New**, and then click **Project**.
3. In the **Installed Templates** section, expand the **Visual C#** node, expand the **SharePoint** node and then click **2010**.
4. Click **Empty SharePoint Project**.
5. In the **Name** text box, type **Lab11**.
6. In the **Location** text box, type **E:\Student\Lab11\Starter** and click **OK**.

   The Microsoft® SharePoint® Customization Wizard appears.
7. In the **What local site do you want to use for debugging?** text box, type **http://sharepoint**.
8. In the **What is the trust level for this SharePoint** solution section, click the **Deploy as a sandboxed solution** option.
9. Click **Finish**.

   The project is created.

▶ Task 2: Add Artifacts to the project

1. In the Solution Explorer window, right-click **Lab11**, click **Add**, and then click **New Item**.
2. Click **Visual Web Part**.
3. In the **Name** text box, type **TestSB_VWP** and then click **Add**.
4. In the Solution Explorer window, right-click **Lab11**, click **Add**, and then click **New Item**.
5. Click **Application Page**.
6. In the **Name** text box, type **TestSB_AppPage.aspx** and then click **Add**.
7. In the Solution Explorer window, right-click **Lab11** and then click **Deploy**.

   Note that the project is not built and deployed because user controls and application pages are not allowed in sandboxed solutions.
8. In the Solution Explorer window, right-click **TestSB_VWP**, click **Delete**, and then click **OK**.
9. In the Solution Explorer window, right-click **Layouts**, click **Delete**, and then click **OK**.

▶ Task 3: Add a Web Part to the project

1. In the Solution Explorer window, right-click **Lab11**, click **Add**, and then click **New Item**.

2. Click **Web Part**.
3. In the **Name** text box, type **BonnevilleTestBed** and then click **Add**.
4. Add the following code to the empty line above the namespace declaration:

```
using Microsoft.SharePoint.Administration;
```

5. Place the cursor after the opening brace of the **public class BonnevilleTestBed** class declaration and press ENTER to create a new line.
6. In the space you have just created, add the following code:

```
LiteralControl output = new LiteralControl();
```

7. Add the following code to the **CreateChildControls** method:

```
SPFarm thisFarm = SPFarm.Local;
output.Text = thisFarm.Name;
this.Controls.Add(output);
```

8. On the **File** menu, click **Save All**.

Task 4: Deploy and test the sandboxed solution

1. In the Solution Explorer window, right-click **Lab11** and then click **Deploy**.
2. Use Windows® Internet Explorer® to browse to **http://sharepoint**.
3. On the **Site Actions** menu, click **Site Settings**.
4. In the **Galleries** section, click **Solutions**.

   The sandboxed solution you have just deployed is listed.
5. In the breadcrumb control, click **Home**.
6. On the ribbon, click **Edit**.
7. On the ribbon, click the **Insert** tab.
8. On the ribbon, click **Web Part**.
9. In the **Categories** section, click **Custom**.
10. In the **Web Parts** section, click **BonnevilleTestBed**.
11. Click **Add**.
12. On the ribbon, click **Save & Close**.
13. Read the error that is displayed in the Web Part. This error is displayed because code in the Web Part accesses the prohibited **SPFarm** object.
14. On the **Site Actions** menu, click **Site Settings**.
15. In the **Galleries** section, click **Solutions**.
16. Point to **Lab11** and then click the drop-down arrow that appears.
17. Click **Deactivate**.
18. In the **Solution Gallery - Deactivate Solution** dialog box, click **Deactivate**.
19. In the breadcrumb control, click **Home**.

20. Read the error that is displayed in the Web Part. This error is displayed because the solution has been deactivated.
21. Leave Internet Explorer running.

Exercise 2: Investigating Allowed and Disallowed Operations in Sandboxed Solutions

▶ Task 1: Complete the Web Part

1. Switch to Microsoft Visual Studio®.
2. Select the three lines of code you previously added to the **CreateChildControls** method.
3. On the toolbar, click **Comment out the selected lines**.
4. Add the following function after the closing brace of the **CreateChildControls** method:

```
void renderWebInfo_Click(object sender, EventArgs e)
{
  try
  {
    SPWeb thisWeb = SPContext.Current.Web;
    string message = string.Format(
      "This web contains {0} subwebs", thisWeb.Webs.Count);
    output.Text = message;
  }
  catch (Exception ex)
  {
    output.Text = ex.Message;
  }
}
```

5. Add the following functions after the function you have just added:

```
void renderWebInfoElevated_Click(object sender, EventArgs e)
{
  try
  {
    SPSecurity.RunWithElevatedPrivileges(showWebCount);
  }
  catch (Exception ex)
  {
    output.Text = "Error caught by my code: " + ex.Message;
  }
}

void showWebCount()
{
  SPWeb thisWeb = SPContext.Current.Web;
  string message = string.Format(
    "This web contains {0} subwebs", thisWeb.Webs.Count);
  output.Text = message;
}
```

6. Add the following function after the functions you have just added:

```
void accessProhibitedNamespace_Click(object sender, EventArgs e)
{
  try
  {
    System.Net.HttpWebRequest.Create(
      "http://intranet.contoso.com");
    output.Text = "Success";
  }
  catch (System.Security.SecurityException secEx)
  {
    output.Text = "Security Violation! Error caught by my code: "
```

```
      + secEx.Message;
  }
  catch (Exception ex)
  {
    output.Text = "Generic Exception. Error caught by my code: "
      + ex.Message;
  }
}
```

7. Add the following code to the **CreateChildControls** method:

```
Button renderWebInfo = new Button();
renderWebInfo.Text = "Access data in this site";
renderWebInfo.Click += new EventHandler(renderWebInfo_Click);
Button renderWebInfoElevated = new Button();
renderWebInfoElevated.Text = "Use elevated privileges";
renderWebInfoElevated.Click
  += new EventHandler(renderWebInfoElevated_Click);
Button accessProhibitedNamespace = new Button();
accessProhibitedNamespace.Text = "Create an HTTP connection ";
accessProhibitedNamespace.Click
  += new EventHandler(accessProhibitedNamespace_Click);
this.Controls.Add(output);
this.Controls.Add(new LiteralControl("<br />"));
this.Controls.Add(renderWebInfo);
this.Controls.Add(renderWebInfoElevated);
this.Controls.Add(accessProhibitedNamespace);
```

8. On the **File** menu, click **Save All**.

Task 2: Deploy and test the sandboxed Web Part

1. In the Solution Explorer window, right-click **Lab11** and then click **Deploy**.
2. Switch to Internet Explorer.
3. On the **Site Actions** menu, click **Site Settings**.
4. In the **Galleries** section, click **Solutions**.

   The Lab 11 solution has been re-deployed and activated by Visual Studio.
5. In the breadcrumb control, click **Home**.

   Three buttons are displayed in the Web Part.
6. Click **Access data in this site**.
7. Read the text displayed in the label above the button. The code has run successfully in the sandboxed Web Part.
8. Click **Use Elevated Privileges**.

   Read the error that is displayed in the Web Part. This error is displayed because code in the Web Part has attempted to perform a prohibited operation by running code with elevated privileges.
9. In the breadcrumb control, click **Home** to refresh the page.
10. Click **Create an HTTP connection**.
11. Read the error that is displayed in the Web Part. Note that this error message has been provided by the code you added, so your exception handler has caught this security violation in a sandboxed Web Part.
12. Close all applications. You have now completed this lab.

Module 12

Lab Answer Key: Working with SharePoint Server Profiles and Taxonomy APIs

Contents:

Lab: Working with User Profiles and Taxonomies Programmatically

Log On to the Virtual Machine for This Lab

For this lab, you use the available virtual machine environment. Before you begin the lab, you must log on to the **10175-LON-DEV-12** virtual machine as **Administrator** using the password **P@ssw0rd**.

Exercise 1: Managing User Profiles

▶ Task 1: Manage user profile properties

1. On the **Start** menu, click **All Programs**, click **Microsoft SharePoint 2010 Products**, and then click **SharePoint 2010 Central Administration**.
2. In the **Application Management** section, click **Manage service applications**.
3. Click **User Profile Service Application**.
4. In the **People** section, click **Manage User Properties**.
5. In the **Contact Information** section, point to **Work e-mail**, and then click the drop-down arrow that appears.
6. Click **Edit**.
7. In the **Edit Settings** section, click **Allow users to edit values for this property**.
8. In the **Display Settings** section, select the **Show in the profile properties section of the user's profile page** check box, and then click **OK**.
9. Close Windows® Internet Explorer®.

▶ Task 2: Create user profiles

1. Start Internet Explorer, and browse to **http://sharepoint**.
2. In the top right section of the page, click the **SHAREPOINT\administrator** drop-down menu, and then click **Sign in as a Different User**.
3. In the **User Name** text box, type **SHAREPOINT\KrishnaS**.
4. In the **Password** text box, type **P@ssw0rd**, and then click **OK**.
5. In the top right section of the page, click the **SHAREPOINT\krishnas** drop-down menu, and then click **My Profile**.
6. Below the picture, click **Edit My Profile**.
7. In the **Work e-mail** text box, type **krishnas@sharepoint.com**.
8. Click **Save and Close**.
9. Near the top of the page, click **My Content**.

   The personal site is created for Krishna.
10. Close Internet Explorer.
11. Start Internet Explorer, and browse to **http://sharepoint**.
12. In the top right section of the page, click the **SHAREPOINT\administrator** drop-down menu, and then click **Sign in as Different User**.

13. In the **User Name** text box, type **SHAREPOINT\MartinR**.
14. In the **Password** text box, type **P@ssw0rd**, and then click **OK**.
15. In the top right section of the page, click the **SHAREPOINT\martinr** drop-down menu, and then click **My Profile**.
16. Below the picture, click **Edit My Profile**.
17. In the **Work e-mail** text box, type **martinr@sharepoint.com**.
18. Click **Save and Close**.
19. Close Internet Explorer.
20. Start Internet Explorer, and browse to **http://sharepoint**.
21. In the top right section of the page, click the **SHAREPOINT\administrator** drop-down menu, and then click **Sign in as Different User**.
22. In the **User Name** text box, type **SHAREPOINT\BennoK**.
23. In the **Password** text box, type **P@ssw0rd**, and then click **OK**.
24. In the top right section of the page, click t**h**e **SHAREPOINT\bennok** drop-down menu, and then click **My Profile**.
25. Below the picture, click **Edit My Profile**.
26. Click **Save and Close**.
27. Close Internet Explorer.
28. Start Internet Explorer, and browse to **http://sharepoint**.
29. In the top right section of the page, click the **SHAREPOINT\administrator** drop-down menu, and then click **Sign in as Different User**.
30. In the **User Name** text box, type **SHAREPOINT\LauraG**.
31. In the **Password** text box, type **P@ssw0rd**, and then click **OK**.
32. In the top right section of the page, click the **SHAREPOINT\laurag** drop-down menu, and then click **My Profile**.
33. Below the picture, click **Edit My Profile**.
34. Click **Save and Close**.
35. Close Internet Explorer.

Exercise 2: Working with User Profiles Programmatically

Task 1: Create an Empty SharePoint Project

1. On the **Start** menu, click **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.

   Microsoft® Visual Studio® 2010 opens.

2. On the **File** menu, point to **New**, and then click **Project**.
3. In the **Installed Templates** section, expand the **Visual C#** node, expand the **SharePoint** node, and then click **2010**.
4. Click **Empty SharePoint Project**.
5. In the **Name** text box, type **Lab12**.
6. In the **Location** text box, type **E:\Student\Lab12\Starter**, and then click **OK**.

   The Microsoft SharePoint® Customization Wizard appears.

7. In the **What local site do you want to use for debugging?** text box, type **http://sharepoint**.
8. In the **What is the trust level for this SharePoint solution** section, click the **Deploy as a farm solution** option.
9. Click **Finish**.

   The project is created.

Task 2: Create an application page for working with user profiles

1. In the Solution Explorer window, right-click **Lab12**, point to **Add**, and then click **New Item**.
2. Click **Application Page**.
3. In the **Name** text box, type **ProfileReporter**, and then click **Add**.
4. Add the following markup between the opening and closing tags of the **<asp:Content>** element that has an **ID** of **Main**:

```
<asp:Label ID="Label1" runat="server"
  Text="Profiles and My Sites"></asp:Label>
<asp:Panel ID="configuredMySites" runat="server" BorderColor="Blue"
BorderStyle="Dashed" BorderWidth="1">
  <asp:Label ID="configuredLabel" runat="server"
  Text="Profiles, Emails, and Personal Sites"></asp:Label>
  <br/>
  <asp:Table ID="configuredTable" runat="server">
  </asp:Table>
</asp:Panel>
```

5. On the **View** menu, click **Code**.
6. In the Solution Explorer window, right-click **Lab12**, and then click **Add Reference**.
7. Click the **.NET** tab.
8. Click **Microsoft.Office.Server**, hold down the CTRL key, and click **Microsoft.Office.Server.UserProfiles**. Click **OK**.

Task 3: Create the application page

1. Add the following **using** statements to directly above the namespace declaration:

```
using System.Web.UI;
using System.Web.UI.WebControls;
using Microsoft.Office.Server;
using Microsoft.Office.Server.UserProfiles;
```

2. Add the following code directly above the **Page_Load** method:

```
UserProfileManager uMan=null;
```

3. Add the following code directly after the closing brace of the **Page_Load** method:

```
void emailUpdater_Click(object sender, EventArgs e)
{
  try
  {
    Button src = (Button)sender;
    UserProfile userProfile =
    (UserProfile)uMan.GetUserProfile(long.Parse(src.CommandName));
    string accountName =
      userProfile["AccountName"].Value.ToString();
    int iPos = accountName.IndexOf("\\");
    accountName = accountName.Substring(iPos + 1);
    string updatedEmail = accountName + "@sharepoint.com";
    userProfile["WorkEmail"].Value = updatedEmail;
    userProfile.Commit();
  }
  catch (Exception ex)
  {
    this.Page.Response.Write(ex.Message);
  }
}
```

4. Add the following code directly after the method you have just added:

```
void mySiteCreator_Click(object sender, EventArgs e)
{
  try
  {
    Button src = (Button)sender;
    UserProfile userProfile =
    (UserProfile)uMan.GetUserProfile(long.Parse(src.CommandName));
    userProfile.CreatePersonalSite();
  }
  catch (Exception ex)
  {
    this.Page.Response.Write(ex.Message);
  }
}
```

5. Add the following code to the **Page_Load** method:

```
try
{
  SPServiceContext svrCtx = SPServiceContext.Current;
  uMan = new UserProfileManager(svrCtx);
  TableCell cell;
  TableRow row;
  row = new TableRow();
  cell = new TableCell();
  cell.Text = "User";
  cell.Font.Bold = true;
  row.Cells.Add(cell);
```

```
cell = new TableCell();
cell.Text = "Email";
cell.Font.Bold = true;
row.Cells.Add(cell);
cell = new TableCell();
cell.Text = "Personal Site";
cell.Font.Bold = true;
row.Cells.Add(cell);
configuredTable.Rows.Add(row);
foreach (UserProfile uProfile in uMan)
{
  ProfileSubtypePropertyManager propMan =
    uMan.DefaultProfileSubtypeProperties;
  string accountName = uProfile["AccountName"].Value.ToString();
  string emailAddr = string.Empty;
  string personalSite = string.Empty;
  if (uProfile["WorkEmail"].Value != null)
  {
    emailAddr = uProfile["WorkEmail"].Value.ToString();
  }
  if (uProfile["PersonalSpace"].Value != null)
  {
    personalSite = uProfile["PersonalSpace"].Value.ToString();
   }
  row = new TableRow();
  cell = new TableCell();
  cell.Text = accountName;
  row.Cells.Add(cell);
  cell = new TableCell();
  if (emailAddr != string.Empty)
  {
    HyperLink emailHyper = new HyperLink();
    emailHyper.Text = emailAddr;
    emailHyper.NavigateUrl = "mailto:" + emailAddr;
    cell.Controls.Add(emailHyper);
  }
  else
  {
    if (uProfile["AccountName"].Value.ToString().ToLower()
      ==SPContext.Current.Web.CurrentUser.LoginName.ToLower())
    {
      Button emailUpdater = new Button();
      emailUpdater.CommandName = uProfile.RecordId.ToString();
      emailUpdater.Text = "Set to default...";
      emailUpdater.Click +=
        new EventHandler(emailUpdater_Click);
      cell.Controls.Add(emailUpdater);
    }
    else
    {
      cell.Text = "Email not set";
    }
  }
  row.Cells.Add(cell);
  cell = new TableCell();
  if ((personalSite == "http://")
    || (personalSite == string.Empty))
  {
    if (uProfile["AccountName"].Value.ToString().ToLower()
      == SPContext.Current.Web.CurrentUser.LoginName.ToLower())
    {
      Button mySiteCreator = new Button();
      mySiteCreator.CommandName = uProfile.RecordId.ToString();
      mySiteCreator.Text = "Create...";
```

```
        mySiteCreator.Click
          += new EventHandler(mySiteCreator_Click);
        cell.Controls.Add(mySiteCreator);
      }
      else
      {
        cell.Text = "Personal site not created";
      }
    }
    else
    {
      cell.Text = personalSite;
    }
    row.Cells.Add(cell);
    configuredTable.Rows.Add(row);
  }
}
catch (Exception ex)
{
  this.Page.Response.Write(ex.Message);
}
```

6. In the Solution Explorer window, right-click **Lab12**, click **Add**, and then click **Add New Item**.
7. Click **Empty Element**.
8. In the **Name** text box, type **ProfileReport**, and then click **Add**.
9. Between the start and end tags of the **<Elements>** element, add the following markup:

```
<CustomAction Id="ProfileActionsMenu"
  GroupId="SiteActions"
  Location="Microsoft.SharePoint.StandardMenu"
  Sequence="1973"
  Title="Profile Reports">
    <UrlAction Url="/_layouts/Lab12/ProfileReporter.aspx" />
</CustomAction>
<CustomAction Id="TaxonomyActionsMenu"
  GroupId="SiteActions"
  Location="Microsoft.SharePoint.StandardMenu"
  Sequence="1981"
  Title="Taxonomy Reports">
    <UrlAction Url="/_layouts/Lab12/TaxonomyReporter.aspx" />
</CustomAction>
```

10. In the Solution Explorer window, right-click **Lab12**, and then click **Deploy**.

▶ Task 4: Test the solution

1. Start Internet Explorer, and browse to **http://sharepoint**.
2. In the top right section of the page, click the **SHAREPOINT\administrator** drop-down menu, and then click **Sign in as Different User**.
3. In the **User Name** text box, type **SHAREPOINT\KrishnaS**.
4. In the **Password** text box, type **P@ssw0rd**, and then click **OK**.
5. On the **Site Actions** menu, click **Profile Reports**.

   Note that the Email and Personal Site columns have values for Krishna.
6. In the top right section of the page, click the **SHAREPOINT\krishnas** drop-down menu, and then click **Sign in as Different User**.

7. In the **User Name** text box, type **SHAREPOINT\MartinR**.
8. In the **Password** text box, type **P@ssw0rd**, and then click **OK**.

   Note that the Email column has a value for Martin, but that he has not yet created his personal site.
9. Click **Create**.
10. On the **Site Actions** menu, click **Profile Reports** to refresh the page.

    Note that the Email and Personal Site columns have values for Martin.
11. In the top right section of the page, click the **SHAREPOINT\martinr** drop-down menu, and then click **Sign in as Different User**.
12. In the **User Name** text box, type **SHAREPOINT\LauraG**.
13. In the **Password** text box, type **P@ssw0rd**, and then click **OK**.

    Note that Laura has not yet set her e-mail address or created her personal site.
14. Click **Set to default**.
15. In the top right section of the page, click the **SHAREPOINT\laurag** drop-down menu, and then click **My Profile**.
16. Below the picture, click **Edit My Profile**.

    Note that Laura's **Work Email** property has now been set.
17. Browse to **http://sharepoint**.
18. On the **Site Actions** menu, click **Profile Reports**.
19. Click **Create**.
20. On the **Site Actions** menu, click **Profile Reports** to refresh the page.

    Note that the Email and Personal Site columns now both have values for Laura.
21. Close Internet Explorer.

Module 13

Lab Answer Key: Developing Content Management Solutions

Contents:

Lab: Branding SharePoint Sites

Log On to the Virtual Machine for This Lab

For this lab, you use the available virtual machine environment. Before you begin the lab, you must log on to the **10175-LON-DEV-13** virtual machine as **Administrator** using the password **P@ssw0rd**.

Exercise 1: Customizing Master Pages

▶ Task 1: Edit a SharePoint site by using SharePoint Designer 2010

1. Start Windows® Internet Explorer®, and browse to **http://sharepoint**.
2. On the **Site Actions** menu, click **Edit Site in SharePoint Designer**.
3. In the Site Objects pane, click **Master Pages**.
4. Right-click **v4.master** and then click **Copy**.
5. Press CTRL+V to paste a copy of the **v4.master** file.
6. Right-click **v4_copy(1).master** and then click **Rename**.
7. Rename the file to **NewMaster.master**.
8. Click **NewMaster.master**.

▶ Task 2: Customize and apply a master page to a SharePoint site

1. In the **Customization** section, click **Edit file**.

   The NewMaster.master file is opened in Microsoft® SharePoint® Designer.
2. Click the **Design** tab.
3. Near the bottom of the **Quick Launch** bar, click **All Site Content** and then press **DELETE**.
4. In the **Quick Launch** bar, click **Libraries** and then on the toolbar, click **Align Text Right**.
5. Click **Save** and then click **Yes**.
6. In the Site Objects pane, click **Master Pages**.
7. Right-click **NewMaster.master** and then click **Set as Default Master Page**.
8. Close SharePoint Designer.
9. Switch to Internet Explorer and refresh the page.

   Your customizations have been applied.

Exercise 2: Applying a Theme to a SharePoint Site

Task 1: Create a theme by using Microsoft PowerPoint 2010

1. On the **Start** menu, click **All Programs**, click **Microsoft Office**, and then click **Microsoft PowerPoint 2010**.

   Microsoft Office Powerpoint® 2010 opens.

2. On the ribbon, click the **Design** tab.
3. Click the **More** drop-down arrow in the **Themes** section, and then click the theme at the end of the third row. (Hint: The tooltip text for the theme is **Metro**.)
4. On the **File** menu, click **Save As**.
5. In the **Save as type** list, click **Office Theme**.
6. In the left-hand pane, click **Desktop**.
7. In the **File name** text box, type **Metro**.
8. Click **Save**.
9. Close PowerPoint and do not save changes if prompted.

Task 2: Apply the Metro custom theme to the SharePoint site

1. Switch to Internet Explorer and browse to **http://sharepoint**.
2. On the **Site Actions** menu, click **Site Settings**.
3. In the **Galleries** section, click **Themes**.
4. Near the bottom of the page, click **Add New Item**.
5. Click **Browse**.
6. Browse to the **Desktop** folder, and then click **Metro**.
7. Click **Open** and then click **OK**.
8. Click **Save**.
9. On the **Site Actions** menu, click **Site Settings**.
10. In the **Look and Feel** section, click **Site theme**.
11. Click **Metro** and then click **Apply**.
12. Close all applications. You have now completed this lab.